IEEE International Conference on Software Maintenance and Evolution (ICSME 2020), Adelaide, Australia, Sept 27-Oct 3 2020.

Score-Based Automatic Detection and Resolution of Syntactic Ambiguity in Natural Language Requirements

Mohamed Osama1, Aya Zaki-Ismail1, Mohamed Abdelrazek1, John Grundy2, and Amani Ibrahim1
1School of IT Deakin University 2Faculty of IT Monash University
Melbourne, Australia
{mdarweish, mohamed.abdelrazek, amani.ibrahim}@deakin.edu.au, john.grundy@monash.edu

***Abstract*—The quality of a delivered product relies heavily upon the quality of its requirements. Across many disciplines and domains, system and software requirements are mostly specified in natural language (NL). However, natural language is inherently ambiguous and inconsistent. Such intrinsic challenges can lead to misinterpretations and errors that propagate to the subsequent phases of the system development. Pattern-based natural language processing (NLP) techniques have been proposed to detect the ambiguity in requirements specifications. However, such approaches typically address specific cases or patterns and lack the versatility essential to detecting different cases and forms of ambiguity. In this paper, we propose an efficient and versatile automatic syntactic ambiguity detection technique for NL requirements. The proposed technique relies on filtering the possible scored interpretations of a given sentence obtained via Stanford CoreNLP library. In addition, it provides feedback to the user with the possible correct interpretations to resolve the ambiguity. Our approach incorporates four filtering pipelines on the input NL-requirements working in conjunction with the CoreNLP library to provide the most likely possible correct interpretations of a requirement. We evaluated our approach on a suite of datasets of 126 requirements and achieved 65% precision and 99% recall on average.**

***Index Terms*—Requirements specification, Requirements analysis, Quality analysis**

I. Introduction

Requirements Engineering is a critical phase of system development. The most important artefact in this phase is the System Requirements Specification (SRS) document [1]. It provides the basis for an agreement between the client(s) and the development team describing what needs to be built [2]. The SRS document contains a detailed description of the system to be built and its expected behaviour including both functional and non functional requirements [3]. Improper requirements – requirements with defects – lead to an increase in the development time and cost of the system and can be the cause of major operational failures [4]. Defected requirements typically includes quality issues and mistakes causing problems in the consequent phases of system development [5], [6]. The most common quality issues include missing or incomplete, incorrect, inconsistent, and ambiguous requirements [5], [6]. Half of the problems in the requirements are caused by incorrect, ambiguous and poorly written requirements, where ambiguity can be identified as the root cause [5], [6]. In [7], [8], it is indicated that fixing errors in requirements or the early phases of design incurs the least cost. Improper requirements also affect the cost and duration of system maintenance.

The majority of system requirements (79%) are written in natural language, such as English, with only 21% using some kind of (semi-)formal notations [9], such as structured natural language like forms [10], templates [11], or formalized language [12]. However, NL is inherently ambiguous, imprecise, and incomplete [13], [14]. To mitigate the inherent problems associated with the reliance on requirements specified in NL, it is pivotal to ensure the quality of the requirements. Ambiguity is one of the most challenging quality issues that may cause numerous problems in the subsequent phases of the development life-cycle. Ambiguity is commonly defined as a sentence with more than one interpretation by different readers [9]. Different interpretations can translate into bugs e.g., design, functional, logical, performance or user interface bugs, if not detected and resolved at the earliest stage possible [8].

Detecting ambiguity at an early stage would decrease the consumed time and cost of the entire system development. To detect such defects, several manual quality assurance processes are put in place. However, detecting ambiguities manually is a hard process that requires high expertise and domain knowledge in addition to the distraction of quality defects as indicated in [15]. Also, reviews of SRS documents should involve all relevant stakeholders [3], who must read, understand and confirm each requirement. Thus, an automated approach for ambiguity detection and/or resolution is primary to ensuring the robustness and reliability of systems as well as enhancing the efficiency of the development process .

Most research addressing the ambiguity problem adopts one of two main strategies: (1) ambiguity prevention and (2) ambiguity detection and resolution. In the former strategy, the main goal is to forbid ambiguities from manifesting in requirements. This ensures that the requirements are stated using precise structure and vocabularies that have unique meaning. Typical approaches define concrete formats such as controlled languages, templates, or patterns (e.g., EARS [10], ACE [16], etc.). In the later strategy, the ambiguities are not avoided from the beginning, but are allowed to exist in the initial SRS documents; however, later eliminated with the aid of appropriate techniques (e.g., machine learning [17] or rule-based [18], [2]). Most of these approaches alert the user for the presence of ambiguity in a sentence. However, they do not provide possible correction(s). In addition, they mainly focus on specific cases of ambiguity and fail to detect other possible instances.

To overcome such issues, we introduce a Score-Based Ambiguity Detector and Resolver (SBADR). Our approach utilises Stanford CoreNLP to obtain the possible parse trees of each sentence of a given textual requirement. SBADR then analyses the generated parse trees using four filtering pipelines to detect and resolve, through suggesting multiple possible interpretations, the syntactic ambiguities in the requirement. These pipelines target: (1) eliminating redundant parsing trees (meaningless or repeated) and, (2) short listing the recommended interpretations. First, redundant interpretations are eliminated to make sure that only the grammatically correct unique interpretations of the sentence are detected. Then, the interpretations are statistically analysed to provide the most possible likelihood interpretations of the requirement. We focus on syntactic ambiguity since its detection is possible by analysing and favouring the syntactically correct reliable interpretations through highest ones at sentence level. On the other hand, detection of semantic and pragmatic ambiguities requires a deeper-level of analysis of meaning at context and discourse level. We evaluated the SBADR against the most relevant approach "Ambigo" on their data sets achieving a better performance. In addition, we evaluated on more complex scenarios –unhandled by Ambigo– available with widely accepted interpretations for reference.

The key contributions of this paper are:

- providing an effective and efficient automatic detection technique for syntactic ambiguity while not being confined to a specific ambiguity pattern;
- resolving the ambiguity by providing the user with the most likelihood interpretation(s); and
- evaluating SBADR on a suite of datasets containing a total of 126 sentences targeting attachment, co-ordination, and analytical types of syntactic-ambiguity.

The rest of this paper is organized as follows: section II gives insight on the ambiguity problem in natural language. Section III proposes the ambiguity detection and resolution technique. In section IV, we present the evaluation of the recall, precision, and F-measure of the proposed approach. Section V discusses the obtained results and highlights the main threads to validity. Section VI, provides a review of the current state of research within our scope. Section VII concludes the paper and highlights the key future work.

II. Motivation

Ambiguity is the prospect of having more than one interpretation to a given sentence. It is one of the intrinsic problems of natural language that occurs more frequently compared to other types of issues as indicated in the empirical study conducted in [19]. Ambiguities are commonly categorized into four types, namely lexical, syntactic, semantic and pragmatic ambiguities [18], [20], [21], [22].

According to the recent review in [20], each of the ambiguity types has main subdivisions. Figure 1 shows the types of ambiguities alongside the main subdivisions of each type in solid lines and our extended subdivision in dashed lines.

These major types are explained as follows.

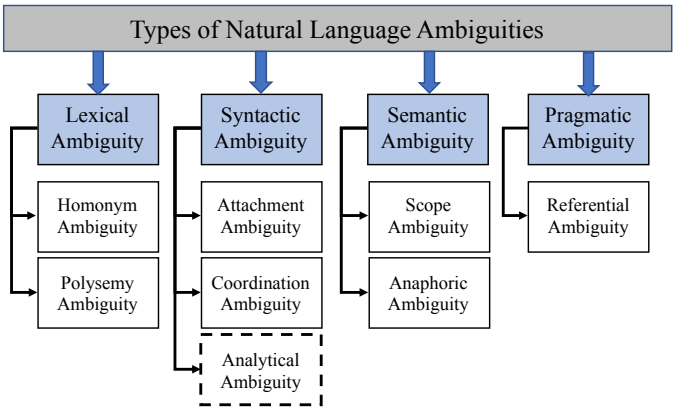


Fig. 1: Types of Ambiguity [20]

Lexical Ambiguity: refers to a situation where a single word has more than one meaning. This type can be further categorized into homonymic and polysemic types of ambiguities. The former type, homonymic ambiguity, occurs when multiple words have the same spellings or phonetics but different meaning as

- Case-1: same spelling with different meanings (e.g., the soldiers liked the port," the word port can refer to a wine or a harbor [23]).
- Case-2: same phonetic, pronounced the same, with different meanings (e.g., Knight and night, The word 'knight', is a person who guards a sovereign or a palace, but the word 'night' is the time period from sunset to sunrise).

While the other type, polysemic ambiguity, occurs when a single word gives different meanings in different contexts (e.g., the word newspaper may mean (1) the physical printed publication as in "The newspaper got wet in the rain" or (2) the publishing institution as in "The newspaper fired some of its editing staff"). From these two examples it can be noticed that polysemy requires the meanings of the word to be etymologically related.

Syntactic Ambiguity: occurs when the whole sentence can have more than one grammatical structure. This type can be divided into Attachment, Coordination and Analytical types of Ambiguity. Attachment ambiguity happens when the under-investigation sentence has suspect of attaching a part of the sentence with another part [19] (e.g., "I saw the boy with the telescope" can be: 1) I saw (the boy with the telescope) or 2) I saw (the boy) with the telescope). Coordination ambiguity appears when a sentence contains coordination or one conjunction is used along with a modifier including two cases:

- Case-1: one conjunction with a modifier (e.g., "young men and women" can be: 1) (young men) and (young women) or 2) (young men) and women)
- Case-2: multi-coordination (e.g., "John and Jack or Marry should come" can be 1) (John and Jack) or (Mary) should come or 2) (John) and (Jack or Mary) should

come).

The last type, Analytical ambiguity, exists when the role of the constituents within a sentence is ambiguous which is common in compound noun phrases (e.g., "general assistance program" can be: 1) (general assistance) program "meaning: program of general assistance" or 2) general (assistance program) "meaning: general program of assistance").

Semantic Ambiguity: in this type of ambiguity, the sentence has several interpretations even though there is neither lexical nor syntactic ambiguities. This type can be decomposed into scope and anaphoric types of ambiguity. Scope ambiguity appears in a sentence with vague quantifying words (e.g., All customers have a reference number may be interpreted as "every customer has a reference number" or "all customers have the same reference number"). Anaphoric ambiguity happens when there are more than one possibility of referring to the word which were mentioned earlier in the sentence [19] (e.g., In their free time, the mothers sing for the children ==> the word their can be referred to by "mothers" or by "children").

Pragmatic Ambiguity: happens when the sentence has more than one meaning within the context [20], [18]. Pragmatic ambiguity can be indicated by referential ambiguity that appears when a word locates its reference from more than one preceding elements [20] (e.g., "the teachers shall provide feedback to students before they go on a holiday" ==> Within the context of the sentence, the word "they" can refer to either the teacher or the students).

In this paper, the main focus is on syntactic ambiguity including coordination, attachment and analytical types of ambiguity. In addition to the key role in correctly understanding software and system requirements (especially in maintenance scenarios), the detection and resolution of syntactic ambiguity is also pivotal across several applications of natural language processing such as Information Extraction, Information Retrieval, and Automatic Speech Recognition [24].

III. Our Approach

An overview of our SBADR tool is shown in Figure 2. SBADR takes a requirements SRS document containing single or multiple sentences per requirement as input. It then separates the sentences of each multi-sentence requirement while keeping their grouping relation intact. Then, each sentence in the requirement is analysed separately to check whether the analyzed sentence is ambiguous or not. If a sentence is found to be ambiguous, the tool marks this sentence within the corresponding requirement as ambiguous and computes the most possible likelihood interpretations. Each interpretation is formatted with grouping brackets to reflect the desired meaning to make easier for the user to understand. Finally, it provides the user with the recommended set of interpretations for each ambiguous sentence detected. The syntactical structures of the suggested interpretations are attached with the sentences.

SBADR utilises the Stanford CoreNLP library as the core underlying technology. Stanford library has been selected for the following reasons:(i) it is the most widely used tool for processing NL requirements [25]; (ii) it is open source with an increasingly growing resources and support community; and (iii) it supports the application of different levels of analysis on NL.

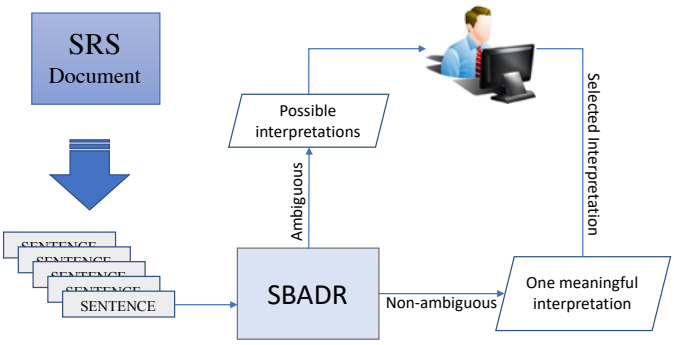


Fig. 2: SBADR Workflow Diagram

A. Framework Description

Algorithm 1 the pseudo-code of the entire ambiguity checking and correction pipeline. Through the utilisation of the Stanford CoreNLP parser, we can parse the sentence into multiple possible interpretations, with each interpretation associated with a parsing confidence score. However, a large number of the obtained interpretations are meaningless (i.e. they do not correspond to a proper English sentence), or repeated. Thus, we crafted a four-filtering pipeline appproach to extract a unique set comprised of the highest scoring meaningful interpretations. If more than one interpretation is obtained, we assist the user in refining the sentence by displaying the possible interpretations.

These four filtering pipelines form the basis of the ambiguity analysis approach. We determined these pipelines by applying extensive analysis on the output of parsing trees obtained by the Stanford NLP parser for several ambiguous and non-ambiguous sentences. A parsing tree is a rooted tree expressing the syntactic structure of the input text with accordance to some grammatical rules of the intended language [26]. The filtering pipelines are as follows:

P1: Meaningless parse trees elimination pipeline: Since the Stanford NLP parser is a statistical-based parser, some of the obtained interpretations may be meaningless. Hence, we check the meaning of each parse tree making sure it conforms to a correct sentence structure as indicated in Figure 3. A sentence structure repository is created based on analysing over 1000 parse trees interpretations for several textual requirements and studying the Stanford CoreNLP documentation. It contains a set of indicators that break the syntactic correctness of a sentence (parse tree starts with fragment, noun phrase, or verb phrase). This elimination is comprehensive within the context of Stanford CoreNLP parsing. The figure shows that, the recommended N scored parsing tree of a given requirement are fed to the filtering pipeline as input. Then, these scored trees are checked against the constructed repository to eliminate any broken interpretation (only parse

Algorithm 1 Ambiguity Checking

Input: Requirement sentence
Output: Interpretation and ParsingTree
procedure GENERATEINTERPRETATIONS(Sentence)
TopTrees ← *Extract the top K-best parsing trees*
Apply four-filtering pipeline
Step1: *ResTrees* ← *eliminate meaningless parsing trees*
Step2: *ResTrees* ← *eliminate redundant parsing trees based on structure*
remove parsing trees with the same structure
Step3: *ResTrees* ← *eliminate redundant parsing trees based on interpretation*
remove parsing trees with the same typed-dependencies
Step4:*ResTrees* ← *keep parsing trees with the highest score within a specific variance*
Within close score variance of the best scoring interpretation "10%"
(found to be an effective ambiguity threshold based on testing done so far)
if $ResTree.size() > 1$ **then**
ParsingTree ← *ask user to select one interpretation*
else
ParsingTree ← *ResTree.get(0)*
end if
Interpretation ← *getInterpretation(ParsingTree)*
return ParsingTree, Interpretation
end procedure=0

trees with roots labeled as "S" or "SBAR" remain and parse trees with internal fragments are removed). Parse tress with containing only noun phrases are also removed (no verb phrase in the sentence). Finally, a set of meaningful interpretations are produced. The P1 column in Fig.7 and Fig.8 show the eliminated meaningless parsing trees in both examples. The examples indicates that, a meaningful interpretation may take a lower score than a meaningless interpretation. This highlights the importance of the filtering pipeline in maximizing chance to correct interpretations with lower scores while avoiding incorrect ones.

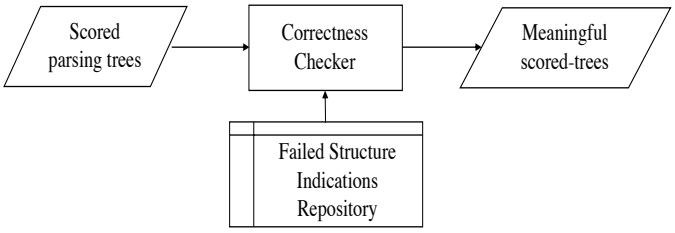


Fig. 3: First filtering pipeline removing meaningless parse trees

P2: Structural-based parse trees filtering pipeline:
In this pipeline, we remove the repeated trees with the same structure. Repeated Parsing trees (based on structure) have three cases: (1) complete identical structure with identical labels, (2) identical structure with interchangeable labels (e.g., NN and NNP) , and (3) identical normalized structures (e.g., NP((DT the)(NN school)) ≡ NP(NP((DT the)(NN school)))). These inconsistencies are resolved in two steps before checking the redundancy as indicated in Fig.4. In S1, interchangeable labels are resolved, first, the parsing trees are flattened and converted to strings. Then, the unlabeled trees' structures are extracted. In S2, the parse trees are normalised, we remove nested redundant braces (e.g., NP inside NP, etc) through bi-directional braces checking (locating corresponding close bracket for each opened bracket). After unifying and normalising the parse trees, we will have a set of parse trees with abstract structures, then we locate the trees with identical abstract structures as indicated in algorithm 2. The P2 column in Fig.7 and Fig.8 show the eliminated parsing trees set with repeated structures. For example, interpretations with indices 1, 7 and 10 in Fig.7 have identical structures with interchangeable labels.

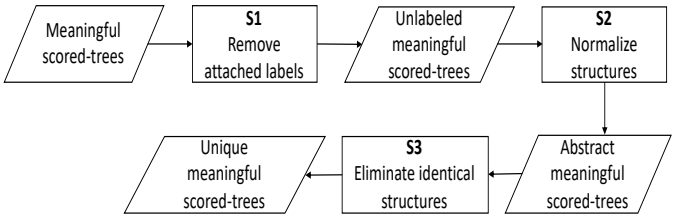


Fig. 4: Second filtering pipeline removing redundant parse trees based on structure

Algorithm 2 Eliminate repeated parsing-trees Case-A

procedure GETUNIQUEPARSINGTREES(ParsingTrees)
TreeStructureList ← ϕ
UniqueParsingTrees ← ϕ
for all crrTree ∈ ParsingTrees **do**
crrStructure ← extract abstract Structure (crrTree)
if crrStructure ∉ TreeStructureList **then**
UniqueParsingTrees ← crrTree
TreeStructureList ← crrStructure
end if
end for
return UniqueParsingTrees
end procedure=0

P3: Redundant interpretations elimination pipeline: For better accuracy, we keep only the interpretations with unique grammatical roles/relations along with unique grammatical structure. Grammatical roles express the semantic relations among words within the input sentence that are known as typed dependencies in Stanford CoreNLP [27]. These relations are extracted from the grammatical structure by identifying the semantic head of each constituent in the parse structure [27]. Thus, typed dependencies of an input sentence are computed

given one parsing tree [27] – the mostly used parsing tree by Stanford is the top scored one. To achieve this kind of filtering, we keep unique parsing trees along with unique typed dependencies as well, as indicated in Figure 5. First, we compute the typed dependency of each parsing tree. Then, for any repeated typed dependency, the corresponding least scoring parsing tree(s) is/are eliminated as in algorithm 3. The P3 column in Fig.8 presents the the eliminated interpretations after applying this type of filtering. The figure visualizes a case of having multiple parsing trees with the same typed dependency. On the other hand, the P3 column in Fig7 shows that the parsing trees set does not change after such type of filter (i.e., means all parsing trees have unique typed dependency).

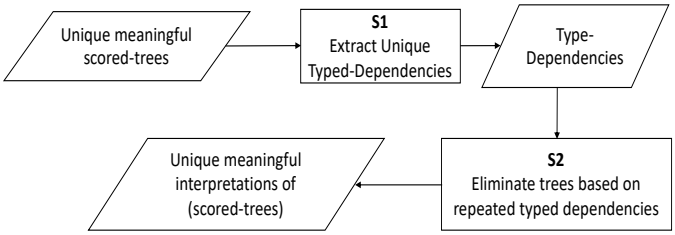


Fig. 5: Third filtering pipeline eliminating redundant parse trees based on typed dependencies

Algorithm 3 Eliminate repeated interpretations Case-B

```
procedure GETUNIQUEPARSINGTREES(ParsingTrees)
    typedDependencyList ← φ
    UniqueParsingTrees ← φ
    for all crrTree ∈ ParsingTrees do
        crrTD ← getTypedDependency (crrTree)
        if crrTD ∉ typedDependencyList then
            UniqueParsingTrees ← crrTree
            typedDependencyList ← crrTD
        end if
    end for
    return UniqueParsingTrees
end procedure=0
```

P4: Score based elimination pipeline: In this level of filtering, we want a concise output set of interpretations reflecting a real case of interpretations. Thus, we favor the more closely related interpretations (score related) over the remaining interpretations as they are less likely to be the intended meaning of the sentence. These related interpretations can be defined based on the attached statistical scores. A score of a given parsing tree expresses the summation of all the decision scores at each node in the parsing tree. A decision score is the selected score ,among the recommended ones each attached with possible semantic, reflecting how plausible the parent node by adopting the semantic [28]. Thus, the relevance degree of two parsing trees' interpretations inversely proportionate to the scores difference of these parsing trees. Consequently, we keep trees within close score with variance $\approx$ 10% (found to be an effective ambiguity threshold based on testing done so far) as indicated in Figure 6. Where the variance would reflect how far the scores are spread out.

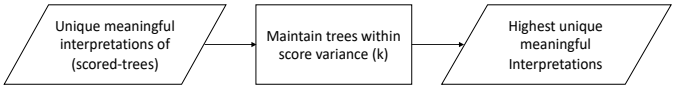
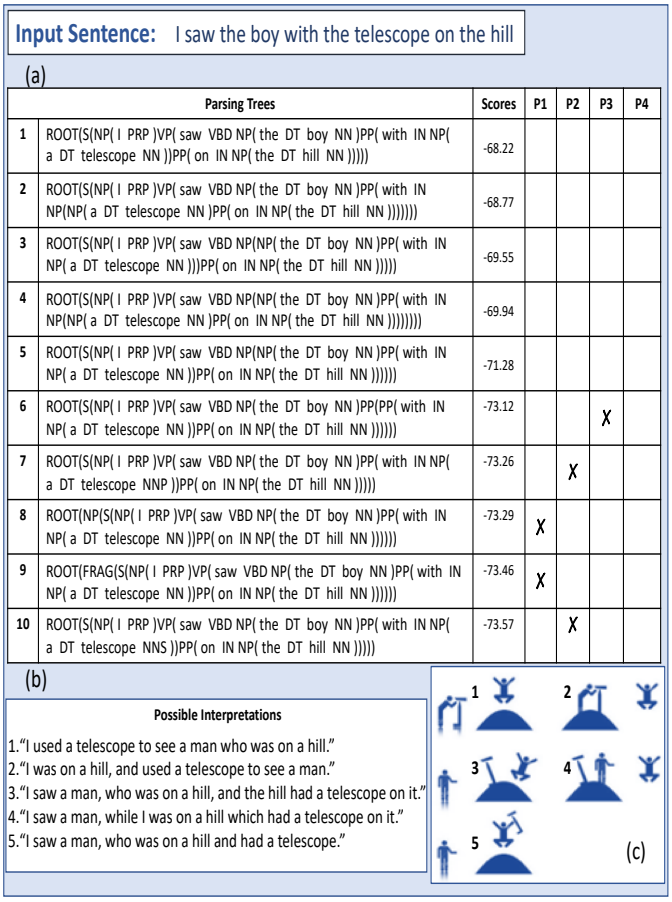


Fig. 6: Fourth Filtering Pipeline



	Parsing Trees	Scores	P1	P2	P3	P4
1	ROOT(S(NP(I PRP)VP(saw VBD NP(the DT boy NN)PP(with IN NP(a DT telescope NN))PP(on IN NP(the DT hill NN)))))	-68.22				
2	ROOT(S(NP(I PRP)VP(saw VBD NP(the DT boy NN)PP(with IN NP(NP(a DT telescope NN)PP(on IN NP(the DT hill NN)))))))	-68.77				
3	ROOT(S(NP(I PRP)VP(saw VBD NP(NP(the DT boy NN)PP(with IN NP(a DT telescope NN)))PP(on IN NP(the DT hill NN)))))	-69.55				
4	ROOT(S(NP(I PRP)VP(saw VBD NP(NP(the DT boy NN)PP(with IN NP(NP(a DT telescope NN)PP(on IN NP(the DT hill NN)))))))	-69.94				
5	ROOT(S(NP(I PRP)VP(saw VBD NP(NP(the DT boy NN)PP(with IN NP(a DT telescope NN))PP(on IN NP(the DT hill NN))))))	-71.28				
6	ROOT(S(NP(I PRP)VP(saw VBD NP(the DT boy NN)PP(PP(with IN NP(a DT telescope NN))PP(on IN NP(the DT hill NN))))))	-73.12			X	
7	ROOT(S(NP(I PRP)VP(saw VBD NP(the DT boy NN)PP(with IN NP(a DT telescope NNP))PP(on IN NP(the DT hill NN)))))	-73.26		X		
8	ROOT(NP(S(NP(I PRP)VP(saw VBD NP(the DT boy NN)PP(with IN NP(a DT telescope NN))PP(on IN NP(the DT hill NN))))))	-73.29	X			
9	ROOT(FRAG(S(NP(I PRP)VP(saw VBD NP(the DT boy NN)PP(with IN NP(a DT telescope NN))PP(on IN NP(the DT hill NN))))))	-73.46	X			
10	ROOT(S(NP(I PRP)VP(saw VBD NP(the DT boy NN)PP(with IN NP(a DT telescope NNS))PP(on IN NP(the DT hill NN)))))	-73.57		X		

Fig. 7: Tracing and investigating the given text: (a) shows the top 10 parse trees, Stanford scores, eliminated trees in P1, P2, P3, P4 respectively from left to right (b) shows the interpretations of the remaining trees ,(c) drawings reflecting each interpretation

Algorithm 4 illustrates the complete filtering steps. First, the candidate parsing trees initialized with the top parsing tree whose score assigned to the mean. In addition, the variance threshold is initialized with 10% of the mean. At each iteration, we extract one parsing tree and calculate the corresponding deviation. In case the square deviation does not exceed the variance threshold, (1) consider this interpretation (2) update mean and variance. The algorithm highlights the continuous update of the interpretations mean and variance at every time of adding a new interpretation and how the update stops.

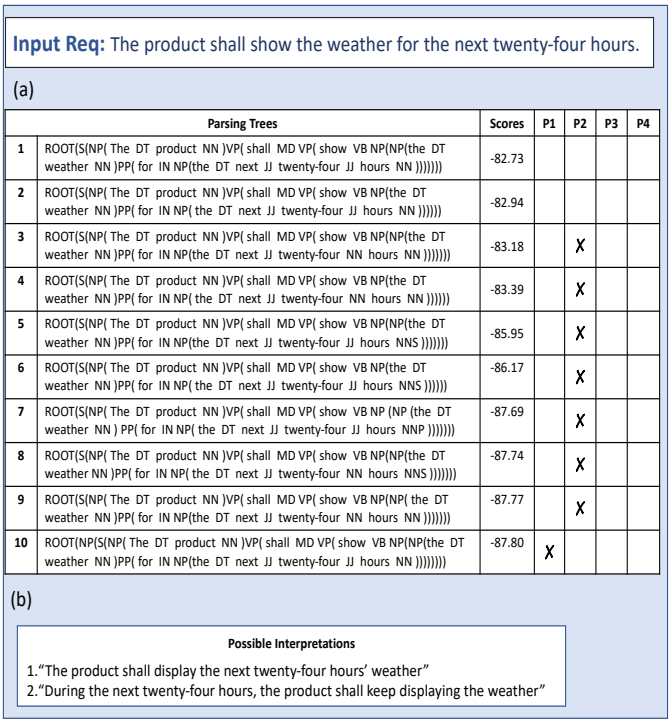


Fig. 8: Tracing and investigating the given text: (a) shows the top 10 parse trees, Stanford scores, eliminated trees in P1, P2, P3, P4 respectively from left to right (b) shows the interpretations of the remaining trees

The point behind updating the variance and not keeping it constant to 10% is adapting the algorithm to the natural deviation of the calculated scores that differs from case to case based on the sentence, the complexity of its structure and the statistical parsing decisions. Consequently, the resulted interpretations will reflect more natural real-cases by striking a balance between comprehensiveness and minimalism. The P4 column in Fig.7 shows the eliminated parse trees after applying such type of filtering. As it can be seen, all the interpretations are in the defined variance. Within which, all of them are unique and correct interpretation as indicated in Figure 7 (b) and (c). Similarly, Fig.8 illustrate the obtained interpretations by applying the four-filtering pipelined approach on NL-requirement. In which, the filter keeps two interpretation 8.(b).

Figure 7 and figure 8 serve as tracing examples for the ambiguity resolution algorithm in different scenarios. The figures illustrate a step by step output highlighting the effect and benefit of each filter. For the input sentence example, the table at (a) presents the top 10 parse trees, the attached Stanford scores, the eleminated parse trees at each filtering pipeline P1, P2, P3 and P4 respectively. (b) and (c) outlines the obtained interpretation(s) after applying the technique and visualise the corresponding meanings respectively. For simplicity, we only extract the top 10 parse trees on a simple sentence. However, in real case scenarios we investigate a maximum of 100 parse trees (for simple sentences, this number of parse tress may not be generated). In the tracing examples (simple sentences) a lower variance percentage could be enough since the decisions made in the scored trees are fewer. However, the recommend percentage is better suited to support a wider range of complicated generic cases without affecting the results of the simpler structures. Figure 7.(b) and figure 8.(b) display the obtained interpretations from our approach for ambiguous and non-ambiguous examples respectively.

Algorithm 4 Keep Highest Scored-Trees

procedure COMPUTE-HIGHESTSCOREDTREES(ParsingTrees)
  varianceRatio = .1
  HighestParsingTrees $\leftarrow \phi$
  scoredParsingTrees $\leftarrow$ attachScores(ParsingTrees)
  topTree $\leftarrow$ scoredParsingTrees.pop()
  meanScore $\leftarrow$ topParsingTree.score()
  scoreSum $\leftarrow$ topParsingTree.score()
  count $\leftarrow$ 1
  threshold = $\leftarrow$ (meanScore* varianceRatio)
  HighestParsingTrees.add(topTree)
  for all crrParsingTree $\in$ scoredParsingTrees **do**
    crrScore $\leftarrow$ crrParsingTree.score()
    diviation = crrScore - meanScore
    if $(\text{diviation})^2$ <= threshold **then**
      HighestParsingTrees.add(crrParsingTree)
      scoreSum $\leftarrow$ scoreSum + crrParsingTree.score()
      count $\leftarrow$ count + 1
      meanScore $\leftarrow$ scoreSum / count
      threshold = computeCurrenVariance
    end if
  end for
  return HighestParsingTrees
end procedure

IV. EVALUATION

In order to establish the effectiveness and efficiency of SBADR in detecting syntactic ambiguity in NL, we conducted two experiments to assess the performance of SBADR against a comparable baseline approach, on a total of 126 sentences with 100 CoreNLP generated interpretations for each sentence (126x100=12600 input interpretations) measuring the recall, precision, and F-measure. **Experiment 1:** evaluates the performance of SBADR against AmbiGO [29] - the most related approach addressing the same types of syntactic ambiguity including: coordination, analytical and attachment ambiguities. **Experiment 2**: evaluates the performance of SBADR on more complex ambiguity cases not handled by the comparable approach. We conducted the experiments using the following settings: (i) number of parse trees undergoing analysis is 100 per sentence, (ii) the initial variance ratio allowed for the remaining top parse trees is 10%, and (iii) the evaluation data sets are comprised of 100 sentences (the original data set used for evaluating AmbiGO) with a total of 10000 input interpretations for experiment 1, and 26 generic sentences curated from multiple sources [30], [31], [32], [33], [34], [35], [36] reflecting common complex and compound cases of ambiguity

for experiment 2. Extensive trials with SBADR have shown that decreasing the value further results in missing possible correct interpretations, while increasing it beyond 10% will only result in more redundant and incorrect parse trees. The datasets, their corresponding automatic output alongside with the manual evaluation and the performance calculation could be found in [1].

A. Experiment 1: SBADR vs AmbiGO

In this experiment, we evaluate our approach against AmbiGO [29]. We selected this approach as it covers the three cases of syntactic ambiguity. We used the same published data sets (testing cases) used to evaluate AmbiGO in [29], which include 50 cases of analytical ambiguity, 28 cases of coordination ambiguity, and 22 cases of attachment ambiguity as categorised by the authors of AmbiGO. It is worth noting that the published test cases only include the detected ambiguous cases, and we manually integrated each of these cases in a sentence form. The supplied cases conform to three POS taggings mentioned in details in [29].

We fed the tool with one data set category at a time to measure the performance for each ambiguity type individually. Table I shows the obtained evaluation measures for each type of ambiguity at the sentence level (measured for each input sentence) including: true positive (TP), false positive (FP), true negative (TN), and false negative (FN) cases in addition to SBADR precision, recall and F-measure. The table also shows these measures at the interpretation level (measured for each generated interpretation).

We then compared the obtained results with AmbiGO results. Figure 9 shows the performance of both tools expressed in (precision, recall and F-measures) for coordination, attachment and analytical ambiguity respectively. SBADR outperforms AmbiGO in recall measures with around 80% or more improvement for all types. This demonstrates that SBADR can be much more reliable compared to AmbiGO in real life scenarios, especially for detecting ambiguities within SRS documents for complex and/or safety critical systems. In such systems, any missed ambiguous requirement, could lead to catastrophic operational consequences or costly fixes in the development cycle. In addition, AmbiGO relies on static predefined rules detecting only specific cases while SBADR do not enforce any templates on the ambiguity detection process. This added flexibility is also expected to make SBADR suited for a range of applications, because the detection process can handle multiple requirements writing styles. On the other hand, AmbiGO considerably outperforms SBADR in the precision for both attachment and analytical types of ambiguity (around 45%, and 35% improvement respectively), while providing around 10% improvement in coordination ambiguity. This is primarily attributed to the fact that AmbiGO incorporates a semantic analysis relying on Google hits count search for each detected case. However, such reliance would not be a reliable approach when applied on technical requirements. The hit count on such terms, would be very low for this analysis to be usable. This is apparent in the fact that in the coordination ambiguity type (where the dependency on the actual meaning is much less compared to the other two types), the improvement provided by AmbiGO is the least. In addition, the precision of SBADR is expected to improve considerably if the input data sets are system requirements with more technical terms, because the relation among most of the involved nouns and noun phrases will be resolved. We also want to highlight an important aspect regarding the way we calculated the precision for SBADR, which is that if a sentence should only have X number of interpretations but SBADR generated Y number of interpretations where Y is greater than X, we count this as false positive. This means that even if the precision of SBADR appears to be lower than AmbiGO (when no limitation is enforced on the number of generated interpretations by SBADR), SBADR can still identify ambiguous sentences with high precision.

Overall, SBADR also has a better F-measure which reflects that the improvement in recall outweigh AmbiGo's lead in precision. This trade off is expected to be in favor of SBADR when utilised for SRS documents. In real life scenarios, loss of precision would require more effort to filter the detected ambiguity cases in a requirements document. On the other hand, the solid recall rate of the utilised approach, is pivotal for the reliability of the ambiguity approach. Any ambiguous requirement that goes undetected could have severe and negative impacts on the system being developed. The cost and effort to filter the detected cases, will be far less than that required to scan the entire requirements documents for undetected cases, which in a typical scenario could be very challenging if done manually, rendering the approach useless. In addition, SBADR provides possible interpretations for each detected case, making the filtering process much easier to go through.

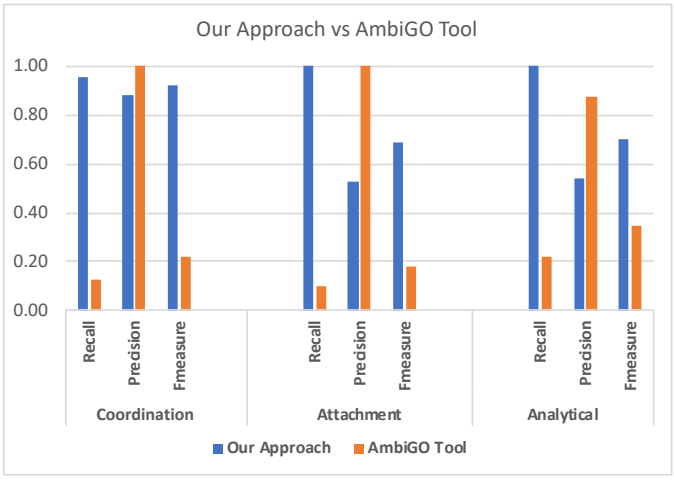


Fig. 9: Performance of SBADR vs AmbiGO [29] on different types of ambiguity

[1]Datasets and Evaluation: https://github.com/ABC-7/NL-Ambiguity

TABLE I: Measured performance of SBADR on AmbiGO data sets

	data-set	TP	FP	TN	FN	Recall	Precision	F-measure
Ambiguity	Coordination	23	3	1	1	96%	88%	92%
	Attachment	10	9	3	0	100%	53%	69%
	Analytical	26	22	2	0	100%	54%	70%
Average						99%	65%	77%
	data-set	**TP**	**FP**	**TN**	**FN**	**Recall**	**Precision**	**F-measure**
Interpretation	Coordination	64	11	0	9	88%	85%	86%
	Attachment	36	28	0	0	100%	56%	72%
	Analytical	120	140	0	0	100%	46%	63%
Average						96%	62%	74%

B. Experiment2: SBADR complex scenarios

In this experiment, we evaluate SBADR on a more complex set of ambiguity scenarios - not addressed by AmbiGO. We aggregated 26 cases –curated from [30], [31], [32], [33], [34], [35], [36]– of complex attachment and co-ordination ambiguity scenarios along with their interpretations to evaluate the reliability of SBADR. Such cases are widely discussed in the field and represent challenging cases of compound and/or mixed instances of ambiguity that are very hard to detect and because of their complexity, specific detection patterns for them can be very difficult to develop and unreliable in practical scenarios.. The curated cases represent the following ambiguity scenarios:

- Chain attachment: presence of more than one prepositional phrase attachment.
- Different attachment types: attachment emerging from non-prepositional cases (e.g., I fed her cat food ==> "I fed her (cat food)" or "I fed (her cat) food"). This is different from the analytical type since the analytical analyses the attachment in one compound noun, but in this case, the ambiguity arises from the attachment of two separate nouns.
- Chain co-ordination: presence of more than one coordination in the sentence.
- Different type coordination: coordination connecting phrases, words, verbs.
- Inverse co-ordination: this one is opposite to the type used in the evaluation because the coordination precedes the modifier (e.g., "I have cheese and tomato sandwiches" ==> "I have (cheese and tomato) sandwiches" or "I have cheese and (tomato sandwiches)".

Table II shows the performance of SBADR for the complex scenarios of attachment and co-ordination ambiguity. The ambiguity column shows the values for each measure based on the ability of SBADR to correctly identify a sentence as ambiguous or not. SBADR correctly identified 6 out of 6 co-ordination cases (TP) achieving 100% for both recall and precision. For the complex attachment scenarios, SBADR correctly identified all the 17 ambiguous cases (TP) and 1 unambiguous case (TN), and missed only 2 ambiguous cases (FN) out of the 20 attachment cases. Upon investigating the missed cases, we found that the parsing from the Stanford CoreNLP library was the reason behind missing these cases. Our tool is still limited by the capabilities of the underlying parser. The interpretation column serves to show the reliability of SBADR in terms of providing a resolution for the detected ambiguity. The reported values reflects the performance based on the evaluation of SBADR on the generated interpretations level (correctness of the interpretations identified for each sentence). If a generated interpretation is valid, it is counted as TP, otherwise, it is counted as FP. Missed interpretations that should have been generated but were missed by SBADR are counted as FN.

TABLE II: Measured performance of SBADR on complex scenarios

	Measures	Ambiguity	Interpretations
Coordination	TP	6	18
	FP	0	4
	TN	0	0
	FN	0	1
	Recall	100%	95%
	Precision	100%	82%
	F-measure	100%	88%
Attachment	TP	17	39
	FP	0	9
	TN	1	0
	FN	2	3
	Recall	90%	93%
	Precision	100%	81%
	F-measure	94%	87%

To assess the reliability of the provided interpretations, we visualized two aspects of the approach performance upon the conducted experiments (AmbiGO data sets and complex scenarios): (1) the number of the produced interpretations per sentence, (2) the correctness of the produced interpretations. First, we grouped the requirements sentences based on the count of the produced interpretations for each sentence. Then, we computed the size percentage of each group relative to the entire set of sentences. Figure 10 shows the identified interpretation groups on the x-axis and the size percentage of each group on the Y-axis. The figure shows that the maximum obtained interpretations count is eight, in addition more than 30% of sentences only have two interpretations and sentences with more interpretations did not exceed 20%. Based on this, users may choose to limit the number of generated interpretations to only two and increase for more complex sentences. We did not limit the number of the generated sentences in our evaluation to gain an insight on the performance of SBADR. We wanted to check that the

generated number of interpretations will remain reasonable and see how the distribution of the sentences interpretation groups looks like (this is expected to differ based on the complexity of the contributing sentences in each data set).

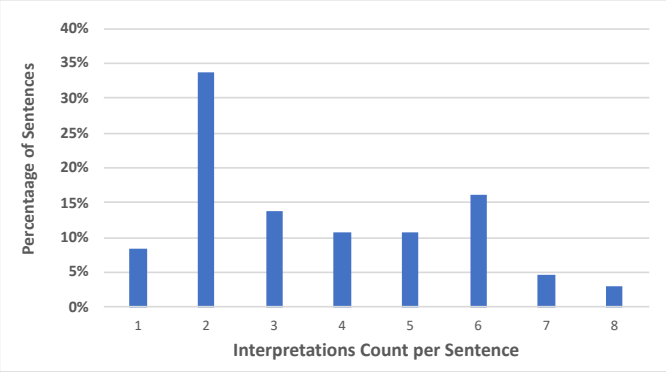


Fig. 10: Percentage of each interpretations count per sentence within the entire generated output

Regarding the correctness, we calculated the percentage of correct interpretations found in each group of sentences (identified based on the number of generated interpretations for each sentence). Figure 11 visualizes the percentage of the correct interpretations found on the y-axis and the corresponding groups on the x-axis. The figure shows that, the correctness of the generated interpretations is inversely proportional to the generated number of interpretations. This is expected as the more possible interpretations are generated, the higher the probability they are not the correct interpretation. Thus, if the number of generated possible interpretations is limited to a maximum of two, the reported precision of SBADR will be greatly improved. We have have not limited SBADR for just two interpretations to assess its performance but it can be easily adjusted based on the user's preference

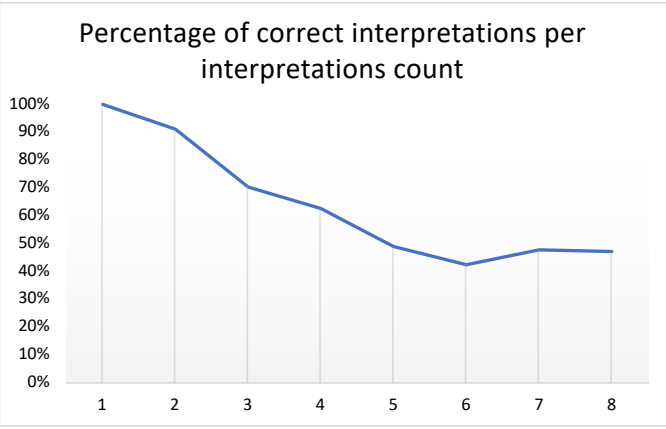


Fig. 11: Percentage of correct interpretations per each interpretation count

C. Threats to Validity

We manually reevaluated the test cases used in the first experiment since several of them were manually labelled wrong. For example, "public utility companies" is marked as non-ambiguous by the author but it is ambiguous, given that the semantic analysis by google indicate the use of the two interpretations but with different frequency". The recommended interpretations may be "public (utility companies)" or "(public utility) companies". Consequently, we recomputed the measure of AmbiGO according to the new manual evaluation –agreed by the authors [1]. Table III shows the changes in the measured performance upon the the new manual evaluation. On the other hand, the manual interpretations of test cases used in the second experiment are widely explained in many NLP resources.

We have applied SBADR on a relatively small set of requirements documents. However we chose a range of domains and used datasets that have previously been used for ambiguity analysis research tool assessment.

An external threat to validity is that the utilised sample dataset might not be representable to other datasets and SRS documents used in a real scenario. The main concern is the size of the dataset used for evaluation. However, the utilised data set for experiment 1 is the one utilised for AmbiGO, and this data set was used to establish the comparison between SBADR and a baseline approach. In addition, this data set was designed specifically to contain several challenging ambiguity instances comparable to what might be found in large SRS documents, given such documents are extensively revised. In addition, regarding how utilising larger data sets might affect the recall rates, we do not expect this to have a negative impact. SBADR detects better than a baseline algorithm developed to detect several cases of ambiguities using set of rules for patterns of ambiguity. The sample data set, albeit small in size, contains several ambiguity patterns that are designed to pose a challenge for detection. Our approach handled these cases successfully. It is also expected to be able to handle other missed patterns as well, since the underlying utilised statistical parser is already trained on millions of sentences and SBADR aims at exploiting that potential by extracting the valid interpretations that, according to the statistical parser scoring, have high chance of being correct. In terms of expected precision, unambiguous sentences in real SRS documents contain technical terms mitigating most of the attachment and analytical issues. In addition, even if a sentence is judged as unambiguous when the semantics is accounted for, the syntactic ambiguity because of the structure of the sentence will still hold and might provide a challenge for automation tools.

V. Discussion

In terms of resolving ambiguous sentences, our approach shows a very good performance by achieving at least 80% precision and recall for the provided interpretations as indicated in Table II. On the other hand, the approach performs less well for the non-ambiguous sentences for three reasons:

- Stanford CoreNLP library utilises a statistical-based parser and may thus assign the top score to a semanti-

cally wrong interpretation (based solely on the syntactic structure) which affects our results.
- The initial variance ratio is static and defined to cover a wide range of cases. Consequently, sentences that require lower variance for a better accuracy are affected.
- A lot of the interpretations are syntactically correct but semantically incorrect due to the lack of semantic analysis.

Our approach provides at most 3 interpretations from the 100 interpretations candidates for more than 50% of the input sentences. This shows the reliability of the approach in supporting users. In addition, the approach rarely misses a viable interpretation and this case mainly happens due to the static initial variance –too small. It is also worth noting that SBADR should benefit from any improvements to the underlying parser utilised. We plan to investigate the effects of utilising other parsers on SBADR. We also plan to further investigate the performance of SBADR on more data-sets with varying sentence complexities.

TABLE III: Old and New Measured performance of AmbiGO Tool

	Measures	Old	New
Coordination	Recall	23%	13%
	Precision	100%	100%
	F-measure	38%	22%
Attachment	Recall	17%	10%
	Precision	100%	100%
	F-measure	29%	18%
Attachment	Recall	15%	22%
	Precision	67%	88%
	F-measure	25%	35%

VI. Related Work

Gleich et al, in [18] developed an automatic ambiguity tool for detecting ambiguity of requirements specifications written in English or German, in addition to educating requirements writers on the potential sources of ambiguity. The tool utilizes natural language processing (NLP) techniques such as: (1) part-of-speech (POS) tagging and (2) regular expressions (RE) . The author built the basis of ambiguity definition and recognition of the proposed ambiguity tool up on two main sources including the Ambiguity Handbook by Berry et al and Siemens-internal guidelines for writing requirements as indicated in [18]. The developed tool is capable of detecting 39 cases of ambiguity with precision and recall scores of 95% and 86% respectively. These cases mostly relate to single word level.

Nigam et al, in [2] have also followed a similar approach in developing an ambiguity detector tool. The tool detects lexical and syntactic ambiguities in software requirement specifications (SRS). All detected cases are based on corpus file, containing words and phrases responsible for ambiguity, fed to the tool. First, each line of the SRS is checked against all lexical words in the corpus to identify ambiguous lines. Ambiguities in a matched line are then classified into semantic, syntactic or syntax ambiguity types by checking with POS tagger. The tool is only capable of detecting ambiguity at the word level and supports the result with some sort of ambiguity percentage statistics in the form of charts and highlighting of the detected ambiguities in text. Precision or recall score of the proposed approach are not provided.

Sabriye et al, in [37] provide a prototype tool called ambiguity detector for detecting syntactic and syntax ambiguity. The tool is very similar to the one proposed by Nigam et al, [2], but it does not provide lexical ambiguity. First, the POS tagger of each sentence is computed (the core of the approach). If a sentence POS does not contain full-stop "." or contains a passive voice it will be marked as a syntax ambiguity. A passive voice in the sentence POS can be detected by matching any of 8 POS patterns proposed by the author. On the other hand, syntactic ambiguity can be detected by checking the inclusion of adjective or adverb. The tool provide chart highlighting the percentage of ambiguous, non-ambiguous, syntactically ambiguous, and semantically ambiguous sentences.

Yang et al, [17] proposed an automatic tool NAI, Nocuous Ambiguity Identification, to identify coordination ambiguities with high risk of misunderstanding among different readers calling it nocuous. The authors define eleven POS-based patterns to extract coordinating ambiguities from natural language requirements document. Then, a machine learning approach is applied to classify the extracted instances as risky or not, subject to a given ambiguity threshold. To effectively predict nocuoucity, collocation frequency heuristics and semantic similarity heuristics are added. NAI tool achieves 70% precision and 100% recall for detection coordination ambiguity with medium user interaction –annotating corpus file with nocuous and innocuous cases for training the machine learning algorithm. for a better accuracy, heuristics need to be developed for allowing different aspects of ambiguity like: (1) chained coordination ambiguity (e.g. “A and B and C”) ,and (2) Ambiguity due the existence of “and/or” combination (e.g. “A and B or C” ==> "A and (B or C)" or “(A and B) or C”).

Olteanu and Moldovan in [24] proposed an automated approach for detecting attachment ambiguity. The provided approach considers the semantic prospective ,utilizing web data as the source of semantic knowledge, as well as the syntactic perspective. The proposed approach uses support vector model whose incorporated features are obtained from the syntactic parsing tree of the candidate requirement sentence, semantic information that are manually annotated in addition to unsupervised information returned from the web. Then, the approach uses the statistical models to predict the frequency ratio of the candidate prepositional phrase attaching the verb or the noun, this way, disambiguate the encountered case. The proposed approach achieved accuracy of 93.62% and 92.85% using Penn Treebank [38] and FrameNet[39] respectively.

Hussain et al., in [15] proposed prototype called requirement specification ambiguity checker (ReqSAC), for automatically assessing textual requirements of SRS documents ,in terms of their ambiguity, by applying text classification system assigning ambiguous and unambiguous flags to each requirement

sentence. The classification technique is based on a Decision tree classifier ,as the size of the training corpus was not large enough to fit neural network training, whose training corpus is manually prepared by the authors. The corpus consists of 1211 sentence ,each marked as ambiguous or non ambiguous, categorized into 165 text passages for 25 different problems. The classifier run on both sentence and discourse levels achieving accuracy of 86.67%.

R Khezri in [29] proposed an automated tool, called AmbiGO, for detecting and resolving syntactic ambiguities in NL text. The purpose of AmbiGO is to assist users with detecting and resolving ambiguities. The tool detects and resolves three types of syntactic ambiguities including: analytical, coordination and PP attachment ambiguities. The author proposes three syntactic-patterns each for one ambiguity type for detecting them. First, the POS of the given sentence is computed, then it is compared with the proposed patterns for detection. Second, AmbiGO augments a semantic analysis to the syntactic one for final ambiguity resolution. It gets the static possible reading corresponding to each pattern. After that, it gets the frequency counts for each possible reading of an ambiguous candidate from Google for semantic analysis. The tool achieved high precision of 100% for coordination and attachment ; and 67% for analytical ,but low recall of 23%, 15%, 17% for coordination, attachment, and analytical types respectively. In addition, the tool is only capable of detecting primitive cases and does not support chain scenarios (e.g., multiple and/or in the coordination, multiple pp attachment, multi-noun entities in the analytical).

Femmer et al, in [3] provide a fully automated tool for detecting quality defects like: slashes, vague adverbs and adjectives, negative words, non-verifiable term, subjective language, imprecise phrase, requirements, comparative requirements, pronouns, Loophole and long sentence. The analysis of the proposed tool is built based on POS tagging and lemmatization with the support of dictionaries. The tool displays warning messages providing details of the detected issues to the user.

Quars [40] and ARM [41] are also automated quality checking tools built up on quality models for assessing quality properties like: ambiguities, incompleteness,.etc by detecting the corresponding language defects called as indicators. These tools process the input requirements to detect lexical and syntactic defects ,using a domain dictionary, for indicators such as vagueness, subjectivity, multiplicity and etc.

Defined formats is one of the traditional approaches used for avoiding ambiguity in requirements. Such formats include: (i) controlled language can be obtained by restricting the grammar and limiting the vocabulary of the natural language [16], [42], (ii) patterns each of them represent the structure of a specific group of requirements [43], and (iii) boilerplates working as parameterized templates for representing requirements with a very limited vocabulary [44]. Many of these formats used for allowing transformation into formal languages. However, they are difficult for all stakeholders and impractical to use defined formats due to the excessive restrictions and limitations following specific patterns of grammar and utilising specific vocabulary as indicated in [20].

VII. Conclusion

In this paper, we proposed an automated approach for ambiguity detection and resolution. The approach detects syntactic ambiguity including three different types: coordination, attachment and analytical ambiguities, at the sentence level using syntactic analysis with the help of Stanford CoreNLP API. The approach is built upon filtering pipelines to eliminate the likelihood of the possible interpretations while increasing the chance of maintaining the higher prospect ones. The approach is capable of providing reasonable and acceptable interpretations to the complex cases that are not handled before (e.g., chain attachment and chain coordination cases). We evaluated SBADR on a total of 126 texting case measuring recall, precision, F-measure at both the sentence and interpretation level.

Although the approach does well in ambiguous scenarios, it achieves a lower performance in the non-ambiguous cases (identifying them as ambiguous) due to the lack of semantic.

We aim to address two main interesting investigations for improving the performance of our tool: (1) augmenting the semantic perspective along side the syntactical one and (2) self learning from the user feedback to avoid other types of redundancy for similar requirements or scenarios.

References

[1] K. Pohl, "The three dimensions of requirements engineering," in *International Conference on Advanced Information Systems Engineering*. Springer, 1993, pp. 275–292.

[2] A. Nigam, N. Arya, B. Nigam, and D. Jain, "Tool for automatic discovery of ambiguity in requirements," *International Journal of Computer Science Issues (IJCSI)*, vol. 9, no. 5, p. 350, 2012.

[3] H. Femmer, D. M. Fernández, S. Wagner, and S. Eder, "Rapid quality assurance with requirements smells," *Journal of Systems and Software*, vol. 123, pp. 190–213, 2017.

[4] D. M. Fernández and S. Wagner, "Naming the pain in requirements engineering: A design for a global family of surveys and first results from germany," *Information and Software Technology*, vol. 57, pp. 616–643, 2015.

[5] I. L. Margarido, J. P. Faria, R. M. Vidal, and M. Vieira, "Classification of defect types in requirements specifications: Literature review, proposal and assessment," in *6th Iberian Conference on Information Systems and Technologies (CISTI 2011)*. IEEE, 2011, pp. 1–6.

[6] P. Chandani and C. Gupta, "An exhaustive requirement analysis approach to estimate risk using requirement defect and execution flow dependency for software development," *Journal of Information Technology Research (JITR)*, vol. 11, no. 2, pp. 68–87, 2018.

[7] G. Mogyorodi, "Requirements-based testing: an overview," in *Proceedings 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems. TOOLS 39*, July 2001, pp. 286–295.

[8] S. Graves. (Oct 2010) Verification validation of flight critical systems (vvfcs) – the way ahead. Fedral Aviation Adminstration.

[9] M. Luisa, F. Mariangela, and N. I. Pierluigi, "Market research for requirements analysis using linguistic tools," *Requirements Engineering*, vol. 9, no. 1, pp. 40–56, 2004.

[10] A. Mavin, P. Wilkinson, A. Harwood, and M. Novak, "Easy approach to requirements syntax (ears)," in *Requirements Engineering Conference, 2009. RE'09. 17th IEEE International*. IEEE, Aug 2009, pp. 317–322.

[11] K. Pohl and C. Rupp, *Requirements Engineering Fundamentals*. Rocky Nook, 2011.

[12] B. Dutertre, M. Sorea *et al.*, "Timed systems in sal," *SRI Int., Menlo Park, CA, USA, Tech. Rep. NASA/CR-2002-211858*, 2004.

[13] L. Lúcio, S. Rahman, C.-H. Cheng, and A. Mavin, "Just formal enough? automated analysis of ears requirements," in *NASA Formal Methods Symposium*. Springer, 2017, pp. 427–434.

[14] R. Fu, X. Bao, and T. Zhao, "Generic safety requirements description templates for the embedded software," in *2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN)*. IEEE, 2017, pp. 1477–1481.

[15] I. Hussain, O. Ormandjieva, and L. Kosseim, "Automatic quality assessment of srs text by means of a decision-tree-based text classifier," in *Seventh International Conference on Quality Software (QSIC 2007)*. IEEE, 2007, pp. 209–218.

[16] N. E. R. S. Fuchs, "Attempto controlled english (ace)," in *CLAW 96, First International Workshop on Controlled Language Applications*, Katholieke Universiteit, Leuven, March 1996.

[17] H. Yang, A. Willis, A. De Roeck, and B. Nuseibeh, "Automatic detection of nocuous coordination ambiguities in natural language requirements," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*. ACM, 2010, pp. 53–62.

[18] B. Gleich, O. Creighton, and L. Kof, "Ambiguity detection: Towards a tool explaining ambiguity sources," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2010, pp. 218–232.

[19] B. Meyer, "On formalism in specifications," in *Program Verification*. Springer, 1993, pp. 155–189.

[20] M. Q. Riaz, W. H. Butt, and S. Rehman, "Automatic detection of ambiguous software requirements: An insight," in *2019 5th International Conference on Information Management (ICIM)*. IEEE, 2019, pp. 1–6.

[21] H. Haron and A. A. A. Ghani, "A survey on ambiguity awareness towards malay system requirement specification (srs) among industrial it practitioners," *Procedia Computer Science*, vol. 72, pp. 261–268, 2015.

[22] A. Ferrari, G. Gori, B. Rosadini, I. Trotta, S. Bacherini, A. Fantechi, and S. Gnesi, "Detecting requirements defects with nlp patterns: an industrial experience in the railway domain," *Empirical Software Engineering*, vol. 23, no. 6, pp. 3684–3733, 2018.

[23] K. BRUCE, "Methods & designs lexical ambiguity of words used in english text," *METHODS*, vol. 1, pp. 1–7, 1978.

[24] M. Olteanu and D. Moldovan, "Pp-attachment disambiguation using large context," in *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2005, pp. 273–280.

[25] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Association for Computational Linguistics (ACL) System Demonstrations*, 2014, pp. 55–60. [Online]. Available: http://www.aclweb.org/anthology/P/P14/P14-5010

[26] D. Jurafsky and J. H. Martin, "Speech and language processing (draft)," *Chapter A: Hidden Markov Models (Draft of September 11, 2018). Retrieved March*, vol. 19, p. 2019, 2018.

[27] M.-C. De Marneffe and C. D. Manning, "The stanford typed dependencies representation," in *Coling 2008: proceedings of the workshop on cross-framework and cross-domain parser evaluation*, 2008, pp. 1–8.

[28] C. Rao and V. N. Gudivada, *Computational Analysis and Understanding of Natural Languages: Principles, Methods and Applications*. Elsevier, 2018.

[29] R. Khezri, "Automated detection of syntactic ambiguity," Master's thesis, Univerzity of Gothenburg, 2017.

[30] Attachment ambiguity. Accessed in Feb, 2020. [Online]. Available: https://languagelog.ldc.upenn.edu/nll/?p=4566

[31] Syntactic ambiguity. Accessed in Feb, 2020. [Online]. Available: https://www.thoughtco.com/syntactic-ambiguity-grammar-1692179

[32] Conjunction ambiguity. Accessed in Feb, 2020. [Online]. Available: http://englicious.org/lesson/conjunctions/conjunctions-conjunctions-and-ambiguity

[33] Studying ambiguous sentences. Accessed in Feb, 2020. [Online]. Available: https://www.byrdseed.com/ambiguous-sentences/

[34] Studying ambiguous sentences. Accessed in Feb, 2020. [Online]. Available: https://www.byrdseed.com/ambiguous-sentences/

[35] Ambiguity types. Accessed in Feb, 2020. [Online]. Available: https://literaryterms.net/ambiguity/

[36] Linguistic ambiguity. Accessed in Feb, 2020. [Online]. Available: https://en.wikipedia.org/wiki/List_of_linguistic_example_sentences

[37] A. O. J. SABRIYE and W. M. N. W. ZAINON, "An approach for detecting syntax and syntactic ambiguity in software requirement specification." *journal of theoretical & applied information technology*, vol. 96, no. 8, 2018.

[38] M. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of english: The penn treebank," 1993.

[39] C. F. Baker, C. J. Fillmore, and J. B. Lowe, "The berkeley framenet project," in *Proceedings of the 17th international conference on Computational linguistics-Volume 1*. Association for Computational Linguistics, 1998, pp. 86–90.

[40] G. Lami, S. Gnesi, F. Fabbrini, M. Fusani, and G. Trentanni, "An automatic tool for the analysis of natural language requirements," *Informe técnico, CNR Information Science and Technology Institute, Pisa, Italia, Setiembre*, 2004.

[41] W. M. Wilson, L. H. Rosenberg, and L. E. Hyatt, "Automated analysis of requirement specifications," in *ICSE*, vol. 97. Citeseer, 1997, pp. 161–171.

[42] V. Sládeková, "Methods used for requirements engineering," Master's thesis, Univerzity Komenského, 2007.

[43] B. Justice, "Natural language specifications for safety-critical systems," Master's thesis, Carl von Ossietzky Universität, 2013.

[44] C. Rupp, *Requirements-Engineering und-Management: professionelle, iterative Anforderungsanalyse für die Praxis*. Hanser Verlag, 2009.