The Design of SREE — A Prototype Potential Ambiguity Finder for Requirements Specifications and Lessons Learned

Sri Fatimah Tjong[1] and Daniel M. Berry[2]

[1] University of Nottingham Malaysia Campus
Jalan Broga, 43500 Semenyih, Selangor Darul Ehsan, Malaysia
nien34@gmail.com

[2] Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, Canada N2L 3G1
dberry@uwaterloo.ca

Abstract. [Context and Motivation] Many a tool for finding ambiguities in natural language (NL) requirements specifications (RSs) is based on a parser and a parts-of-speech identifier, which are inherently imperfect on real NL text. Therefore, any such tool inherently has less than 100% recall. Consequently, running such a tool on a NL RS for a highly critical system does not eliminate the need for a complete manual search for ambiguity in the RS. **[Question/Problem]** Can an ambiguity-finding tool (AFT) be built that has 100% recall on the types of ambiguities that are in the AFT's scope such that a manual search in an RS for ambiguities outside the AFT's scope is significantly easier than a manual search of the RS for *all* ambiguities? **[Principal Ideas/Results]** This paper presents the design of a prototype AFT, SREE (Systemized Requirements Engineering Environment), whose goal is achieving a 100% recall rate for the ambiguities in its scope, even at the cost of a precision rate of less than 100%. The ambiguities that SREE searches for by lexical analysis are the ones whose keyword indicators are found in SREE's ambiguity-indicator corpus that was constructed based on studies of several industrial strength RSs. SREE was run on two of these industrial strength RSs, and the time to do a completely manual search of these RSs is compared to the time to reject the false positives in SREE's output *plus* the time to do a manual search of these RSs for only ambiguities not in SREE's scope. **[Contribution]** SREE does not achieve its goals. However, the time comparison shows that the approach to divide ambiguity finding between an AFT with 100% recall for some types of ambiguity and a manual search for only the other types of ambiguity is promising enough to justify more work to improve the implementation of the approach. Some specific improvement suggestions are offered.

1 Introduction

This paper describes the engineering and design of a prototype ambiguity finding tool (AFT) called SREE for helping mainly requirements analysts (and anyone else with a stake in avoiding ambiguity) to find ambiguities in natural language (NL) requirements specifications (RSs). Berry, Gacitua, Sawyer, and Tjong suggest that to be really useful for application to an RS for a critical system, an AFT must have 100% recall even at the expense of low precision. In order that this high imprecision be manageable, the AFT must produce output that is significantly smaller than the original RS submitted to the AFT, so that searching for false positives in the output is significantly easier than manually searching the whole RS for the same ambiguities [1]. They suggest dividing the process of finding ambiguities in a NL RS into two parts that must be easily distinguished:

1. the algorithmic part that can be done with 100% recall by a tool, the dumb tool, and
2. the nonalgorithmic part that is smaller than manually searching the entire RS and which requires the intelligence of a smart human.

As explained in Section 4.1, by "easily distinguished" is meant that for any *type of ambiguity*, the user knows for sure, in advance, in which part all of the type's instances will be found. While, in the end, SREE did not quite achieve this goal, a description of SREE's design and of the process by which it was designed and evaluated against the goals, are useful for informing any future research done according to Berry *et al*'s research agenda.

1.1 Background

Ambiguity in natural language (NL) is a major problem in many scientific disciplines [2, 3], including in requirements engineering (RE) [4] during which requirements specifications (RSs) for computer-based systems (CBSs) are written. Ambiguity in a requirement statement (RStat) occurs when the RStat has more than one meaning. The overwhelming majority of RSs are written in NL [5], although often amplified by information in other notations, such as formulae and diagrams. Despite NL's inherent ambiguity and informality, NL is still preferred for writing RSs simply because all stakeholders are able to read it and participate in writing it.

For a summary of the types of ambiguities that appear in requirements documents, please consult the tutorial titled "Ambiguity in Requirements Specification" [6]. Linguists consider ambiguity, imprecision, indeterminacy, and vagueness to be different phenomena. Nevertheless, the work described in this paper collapses all of these phenomena into one term "ambiguity", because all the phenomena have the same effect on RSs, making them interpretable differently by different developers. An ambiguous RStat can have hazardous consequences for a software development project as a whole, in which wrongly implemented requirements cause high costs for rework and delayed product releases [2–4].

In understanding a RStat, a requirements analyst often is not even aware of any ambiguity and instinctively employs immediate *subconscious disambiguation* [4]. In subconscious disambiguation, the listener or reader of an utterance, not even aware of the existence of ambiguity in the utterance, accepts as the only meaning of the utterance, the first meaning he thought of, a meaning that may not be that intended by the speaker of the utterance.

An AFT's *recall* is the percentage of the instances of ambiguity that the AFT actually finds. The AFT's *precision* is the percentage of the ambiguities that the AFT finds that are truly ambiguous. It is easy to achieve 100% recall if one does not care about precision: just identify everything as ambiguous. Conversely, it is easy to achieve 100% precision if one does not care about recall: just identify nothing as ambiguous. Neither of these ways of achieving perfection in one goal is useful. Thus, recall and precision are usually traded off. Nevertheless, in any tradeoff for the design of an AFT, recall is more important than precision for the AFT, because it is much harder to know what is missing than to know what is a false positive among the AFT's output.

1.2 This Paper

This work is derived from Tjong's Ph.D. thesis [7], and it describes the design and development of a prototype AFT, SREE (Systemized Requirements Engineering Environment), that helps detect the occurrence of instances of ambiguity in RSs. The performance goal for SREE, reflected in its design, is that it has 100% recall of the ambiguities in its scope even at the cost of it having less than 100% precision. For any type of ambiguity in SREE's scope, SREE searches in its input for all instances of that type in its goal of achieving 100% recall, and it reports all instances it finds. If, however, SREE has less than 100% precision for the type, then among the findings are false positives that the human user must weed out in a manual examination of the findings. Thus, what SREE actually finds are *potential ambiguities*. Therefore, SREE assists and does not replace an analyst in identifying instances of ambiguity in RSs. A *user* of SREE must decide whether or not any potential ambiguity detected by SREE is truly ambiguous.

Section 2 of this paper reviews past work about AFTs and finds all deficient in recall. Section 3 explains why 100% recall is essential for an AFT. Section 4 describes the goals of SREE. Section 5 describes the design and implementation of SREE according to these goals. Section 6 evaluates SREE with respect to its goals and the user's performance in dealing with imprecision, and it reports lessons learned about the functionality of AFTs. Section 7 summarizes the paper and suggests future work.

Quoted text from any RStat or other example is typeset in a sans serif typeface, and this text may end with punctuation, which is to be distinguished from any immediately following punctuation belonging to the sentence containing the quoted text, which is typeset in the background serifed typeface[1].

2 Past Work on Ambiguity Finding Methods and Tools

Research to resolve ambiguity started as early as the late 1980s, at which time ambiguity was said to be an impediment to elucidating a project's real design requirements [4]. This section focuses particularly on tool-assisted ambiguity detection and disambiguation.

Ambriola and Gervasi [8] achieved a high-quality NL RS through successive transformations that are applied to an early NL RS in order to obtain concrete and rendered views of models extracted from the NL RS. The transformations are automated in a tool named CIRCE (Cooperative Interactive Requirement-Centric Environment).

Wilson et al. [9] defined general quality criteria for RSs and developed an analysis tool ARM (Automated Requirements Management) to assess the structure of a given

[1] A good magnifying glass shows the difference in the typefaces of the two consecutive items of punctuation! :-).

RS, the structure of the RS's RStats, the vocabulary used to write the RS, and thus to determine if the RS meets the quality criteria.

Fabbrini et al. [10, 11] distinguished between RStat quality and RS quality, and they identified a number of indicators of each kind of quality. They developed QuARS (Quality Analyser of Requirements Specifications) that evaluates any given RS against these indicators of RStat quality and RS quality.

Kasser [12] developed TIGER Pro, a tool that assists its user in writing a good RS by pointing out any instances of five types of defects it finds in the input RS. The five types of defects are 1) multiple requirements in a paragraph, 2) unverifiable requirements, 3) use of will or must instead of shall, 4) use of a wrong word, and 5) use of any user-defined so-called poor word.

Willis, Chantree, and De Roeck [13] defined a *nocuous ambiguity* as ambiguous text that is interpreted differently by different readers, as opposed to an innocuous ambiguity, which in spite of the ambiguity, is interpreted the same by all readers. Generally, domain knowledge or context allowing disambiguation of an ambiguity renders an ambiguity innocuous. Willis *et al.* developed and evaluated a tool using a heuristic method to automatically predict which sentences are likely to be interpreted differently by different readers. After determining which sentences *are* ambiguous with a parser, the heuristic, which is trained against a corpus in which all nocuous ambiguities have been marked, determines which ambiguous sentences are likely to be nocuously ambiguous.

Kiyavitskaya, Zeni, Mich, and Berry [14] did some case studies with prototypes of a proposed tool for identifying ambiguities in NL RSs in an effort to identify requirements for such tools. Their approach was to apply ambiguity measures to sentences identified by a parser based tool to try to increase the precision of the tool with respect to reporting genuine ambiguities. The measures are based on using lexical and syntactic proxies for semantic criteria and the WordNet thesaurus [15]. The case studies found that many of what the tool thought was ambiguous were not problematic given the normal knowledge that the analyst user would have about the domain of the specification and that the tool failed to find many of what one analyst who was particularly attuned to finding ambiguities found manually.

Gleich, Creighton, and Kof [16] built an automated AFT that has been measured to be about four times as effective as an average human analyst in finding genuine ambiguities. They have designed the AFT to automate the finding of ambiguities in requirements specifications, to make plausible to the user that the ambiguities it has found are genuine, and to teach the analyst by explaining the sources of the ambiguities it has found. The AFT first does part-of-speech (POS) tagging of its input and then uses a combination of techniques including simple keyword matching, regular expressions, and regular expressions matching POS tags to find instances of the ambiguities that are in its domain.

Some of the AFTs focus specifically on coordination ambiguity, that involving the coordinators, such as and and or. For example, Agarwal and Boggess [17] developed a tool that identifies a coordinator by matching the POS and semantic tags of the words modified by the coordinator. Resnik [18] proposed a semantic-similarity approach to disambiguate coordinators that involves nominal compounds. Goldberg [19] developed a tool that applies a co-occurrence-based probabilistic model to determine the attachments of ambiguous coordinators in phrases, using unsupervised learning.

Chantree, Willis, Kilgarriff, and De Roeck [20], in work predating that by Willis *et al.*, developed a binary classifier for coordination ambiguity. The classifier is based on a set of ambiguous phrases from a corpus of requirements specifications and a collection of associated human judgements on their interpretations. The classifier uses word distribution information obtained from the British National Corpus (BNC).

The common drawback of many of these AFTs is that the recall of each is less than 100%, even for the ambiguities it claims to be detecting. Certainly, any AFT that depends on an auxiliary parser to find sentences with multiple parses or that depends on an auxiliary POS tagger to identify the POS of any word is going to have a recall of less than 100%, for no other reason than that no version of any of these auxiliary tools is perfect. Each parser or POS is easily fooled by anomalous sentences and words that can be in more than one POS, e.g., a word that is both a noun and a verb. The next section explains why an AFT's having less than 100% recall is a serious drawback.

On the other hand, each of ARM, QuARS, and TIGER Pro attempts to identify only specific types, namely those in its published scope. Its scope is designed to be some classes of ambiguities which can be recognized completely from keywords.

3 Why AFTs Applied to RSs of Critical Systems Need 100% Recall

Suppose that the CBS that is being built whose RS is being examined for ambiguities is a life-, safety-, or security-critical system. Then it is essential to find *all* ambiguities in the RS in order to resolve them. An AFT that does not find all ambiguities in the RS, because of inherent limitations, provides no real benefit to its user who must search the entire RS manually anyway to find the missing ambiguities. Use of the AFT is then a waste of time, and it may make the user less diligent in her search[2].

The fact that an AFT finds, say, even 90% of the ambiguities is of no particular help, because the user has no idea which 90% the AFT found and cannot just focus on finding the missing 10%, which are nevertheless critical to find. The missing 10% are not in any geographically identifiable parts of the RS; the missing 10% are not of any specifically identifiable types of ambiguity. The missing 10% arise from the fact that the underlying parser, POS tagger, or both are not perfect in unpredictable ways on actual input[3].

It is clear that some types of ambiguities are easier to find algorithmically than others. For example, a coordination ambiguity can occur only in a sentence with at least two occurrences of and or or and is thus easier to find than any semantic ambiguity. The approach taken in the work reported in this paper is to partition ambiguities by type into those that can be found algorithmically and those that cannot. Then an AFT would

[2] Of course, if the AFT is used *after* the human has made a serious attempt to find all ambiguities, then the AFT's output can be used to complement the human's findings. However, we would have to guard against the human's getting sloppy in her search in anticipation of the AFT's later use.

[3] Actually, it can be argued that an AFT needs to be only as good as a human ambiguity finder, who is, in the end, incapable of finding 100% of the ambiguities.

be built with 100% recall for the algorithmically findable types of ambiguities. In fact, even coordination ambiguities cannot be found with complete accuracy, i.e., with 100% recall *and* 100% precision. Some sentences with at least two occurrences of and or or are not coordination ambiguous. If the human user can tolerate imprecision, then identifying coordination ambiguity can be made to have 100% recall by presenting to the user every sentence with at least two occurrences of and or or. Then the user has to look at each presented sentence to decide if indeed it is coordination ambiguous or even ambiguous in some other way. The tradeoffs here are (1) recall versus precision and (2) imprecision versus burden to the user. The set of ambiguity types found by an AFT with 100% recall will be selected by careful consideration of these tradeoffs. This selection is a major focus of the research reported in this paper.

A type of ambiguity is in an AFT's scope if and only if the AFT can achieve 100% recall of instances of the type. The user knows that she can focus on finding instances of only those types of ambiguity outside the AFT's scope with the full confidence that the AFT will find and present as a potential ambiguity every instance of each type of ambiguity in the AFT's scope. It is, therefore, necessary to determine what types of ambiguities are detectable with 100% recall and tolerable imprecision and are, therefore, in the AFT's scope.

A type of ambiguity depends on a keyword if the keyword must be present in every instance of the type of ambiguity. For example, every instance of the only ambiguity must contain the word only. It is possible to have 100% recall of the only ambiguity simply by identifying every sentence with the word only. Of course, not every sentence with only is ambiguous. Therefore, simply identifying every sentence with the word only suffers from less than 100% precision for the only ambiguity. Being more precise about the only ambiguity requires being able to identify any sentence in which its only directly precedes the main verb of the sentence.[4] Doing this identification requires being able to identify the main verb of a sentence, which in turn requires parsing sentences and tagging each word with its POS. We know that parsing and POS tagging cannot be done by software with 100% accuracy. Therefore, achieving greater precision in recognizing instances of the only ambiguity costs a reduction in recall and may still not achieve 100% precision.

The goal of the research reported in this paper is to design an AFT that has 100% recall for all ambiguity types selected to be in its scope even if the 100% recall costs less than 100% precision. The AFT, as software, cannot have the intelligence necessary to determine if a potential ambiguity is a true ambiguity, but a human user has the intelligence. Therefore, the determination whether a potential ambiguity is ambiguous is left to the AFT's user. This division of labor and relaxation of the goal of 100% precision means that the AFT can afford to be less than precise in its goal to identify all possible potential ambiguities. However, if the AFT's imprecision is too high to the point that the user feels that using the AFT is burdensome, the AFT may end up not being used at all.

[4] The only ambiguity stems from the English convention of putting a sentence's only immediately before the sentence's main verb, regardless of what word or phrase is limited by the only. Thus, an only that precedes other than its sentence's main verb is probably not ambiguous.

Any AFT needs to be compared *in the way it is intended to be used* against a fully manual search for the same ambiguities. That is, if an AFT is intended to be used in conjunction with some manual work then that manual work must be considered in the comparison of the AFT with a fully manual search. The recall of the AFT-plus-its-required-manual work is to be compared with the recall of the fully manual search. The same must be done for time-required-for-using and for precision.

4 SREE

The main purpose for designing SREE was to experiment with the particular decomposition of the ambiguity identification task that meets the goals described in Section 3 for an AFT. The version of SREE described in this paper is in fact the third attempt, after two attempts [21–23] using more traditional parser-based designs failed to produce an AFT that was significantly better than the existing AFTs.

4.1 Use Lexical Analyzer Instead of Syntactic Analyzer for AFTs

The syntax analyzer that the Tjong developed in her early work [24] achieved only an 80% recall rate of correctly tagged words without any POS tagging algorithm. With more work, the parser could have been improved with a deep-search-and-match heuristic [25] and with machine learning [26]. However, as mentioned at the end of Section 2, achieving 100% accuracy in parsing and POS tagging is impossible. A SREE based on a parser would not be able to achieve 100% recall of any ambiguity whose recognition depends on having a correct parse or a correct POS assignment.

On the other hand, using a pure lexical (in the sense of compilation) strategy would allow SREE to achieve 100% recall of any potential ambiguity for which *all* of its indicators are specific words, i.e., a type of potential ambiguity that would be in SREE's scope.

We decided that SREE would do only lexical searches for potential ambiguities. Therefore, from a user's viewpoint, there are two types of potential ambiguities:

1. those in the scope of SREE, because they are lexically identifiable with 100% recall and
2. those not in the scope of SREE, because they cannot be identified lexically with 100% recall.

As mentioned, the user knows that SREE searches for only potential ambiguities in its scope. The user knows that she must decide for any potential ambiguity whether it *is* an ambiguity. The user knows that she must search manually for any kind of ambiguity not in SREE's scope. She can do this search while ignoring those types of ambiguities that are in SREE's scope, thus allowing her to focus on what she must find manually. This ability to focus on fewer types of ambiguities may increase her effectiveness in finding instances of the types of ambiguities she is focusing on.

4.2 Research and Design Method

To start the development of SREE in 2007, Tjong gathered a set of industrial-strength RSs [7]. In these RSs, she found all the instances of ambiguity she could. Each instance became the indicator of some potential ambiguity for SREE.

The set of RSs consists of the RSs from two case studies [27] and seven industrial strength RSs [28–34]. Each RS in the set contains both functional RStats and nonfunctional RStats. Tjong then studied each RStat in the RSs and identified all ambiguities she could find in the RStat. She used the ARM and QuARS indicators to help her find ambiguities. Each ambiguity was classified by its ambiguity type, such as coordination ambiguity, weak auxiliary, vagueness, etc.

For each type of potential ambiguity, a list of indicators was built in the hopes of finding an exhaustive list of indicators for the type. For some types of potential ambiguities, e.g., coordination and misplaced only, constructing an exhaustive list seems possible. For others, e.g., plural, it is probably impossible. For each type of potential ambiguity, part of the research is to determine if the type can be part of the scope of SREE, i.e., all of its indicators can be found and be used to build a lexical analyzer for the potential ambiguity with 100% recall. The game to be played is to see if the list of the indicators for a type of potential ambiguity stabilizes, i.e., no new elements for the list are found after some reasonable number of uses of SREE in a domain.

5 Architecture and Construction of SREE

This section describes the architecture and construction of SREE, which permit SREE to be modular, easy to change[5], extensible, and easy to use. Basically, SREE has two main components, the AIC and the lexical analyzer.

5.1 Ambiguity Indicator Corpus (AIC)

The AIC contains the corpus of indicators of potential ambiguity. Because of the vast richness of NL, it is simply not possible to have an AIC that contains an indicator of every possible potential ambiguity. Therefore, SREE allows its user to add new indicators to its AIC. There are two AICs in SREE, the original indicator corpus (OIC) and the customized indicator corpus (CIC). The OIC contains ten subcorpora, each in a separate file, each with its own list of indicators, and each named for the nature of the potential ambiguities indicated by elements of its contents. The indicators in these subcorpora are:

- *Continuance*: contains indicators: as follows, below, following, in addition, in particular, listed, meantime, meanwhile, on one hand, on the other hand, and whereas.
- *Coordinator*: contains the indicators: and, and/or, and or.
- *Directive*: contains the indicators: e.g., etc., figure, for example, i.e., note, and table.
- *Incomplete*: contains the indicators: TBA, TBC, TBD, TBE, TBS, TBR, as a minimum, as defined, as specified, in addition, is defined, no practical limit, not defined, not determined, but not limited to, to be advised, to be defined, to be completed, to be determined, to be resolved, and to be specified.

[5] The prototype's being easy to change is critical when one is continually subjecting the prototype to changes as new requirements are discovered.

- *Optional*: contains the indicators: as desired, at last, either, eventually, if appropriate, if desired, in case of, if necessary, if needed, neither, nor, optionally, otherwise, possibly, probably, and whether.
- *Plural*: contains a list of 11,287 plural nouns, each ending in "s". We differentiate the terms "Pluralnoun" and "plural noun". The former is what is detected by SREE as a result of its use of the Plural corpus. The latter is the collection of nouns, which are of plural types. SREE has 100% recall of Pluralnouns, but not of plural nouns.
- *Pronoun*: contains the indicators: anyone, anybody, anything, everyone, everybody, everything, he, her, hers, herself, him, himself, his, i, it, its, itself, me, mine, most, my, myself, nobody, none, no one, nothing, our, ours, ourselves, she, someone, somebody, something, that, their, theirs, them, themselves, these, they, this, those, us, we, what, whatever, which, whichever, who, whoever, whom, whomever, whose, whosever, you, your, yours, yourself, and yourselves.
- *Quantifier*: contains the indicators: all, any, few, little, many, much, several, and some.
- *Vague*: contains the indicators: /, < >, ( ), [ ], , ;, ?, !, adaptability, additionally, adequate, aggregate, also, ancillary, arbitrary, appropriate, as appropriate, available, as far as, at last, as few as possible, as little as possible, as many as possible, as much as possible, as required, as well as, bad, both, but, but also, but not limited to, capable of, capable to, capability of, capability, common, correctly, consistent, contemporary, convenient, credible, custom, customary, default, definable, easily, easy, effective, efficient, episodic, equitable, equitably, eventually, exist, exists, expeditiously, fast, fair, fairly, finally, frequently, full, general, generic, good, high-level, impartially, infrequently, insignificant, intermediate, interactive, in terms of, less, lightweight, logical, low-level, maximum, minimum, more, mutually-agreed, mutually-exclusive, mutually-inclusive, near, necessary, neutral, not only, only, on the fly, particular, physical, powerful, practical, prompt, provided, quickly, random, recent, regardless of, relevant, respective, robust, routine, sufficiently, sequential, significant, simple, specific, strong, there, there is, transient, transparent, timely, undefinable, understandable, unless, unnecessary, useful, various, and varying.
- *Weak*: contains the indicators: can, could, may, might, ought to, preferred, should, will, and would.

SREE automatically loads these corpora into the AIC each time a user starts up SREE. Note that the indicators for the only ambiguity are already listed among the indicators for vagueness.

The user of SREE is not allowed to modify or delete any of the original corpora in the OIC. He may add to the CIC any indicator of potential ambiguity that he may find that is not in the AIC. He may also remove from the CIC indicators that have proved less than helpful.

5.2 Lexical Analyzer

SREE's lexical analyzer scans a RS, RStat by RStat, and scans each RStat, token by token, for any occurrence of any indicator in the AIC. During the scan, the lexical analyzer of SREE reads tokens from its input RS and compares each token with each indicator in the AIC. If SREE finds a match, it reports the token and its containing RStat as a potentially ambiguous Rstat.

6 Evaluation of Design and Acceptability of Imprecision Amounts

The question that needs to be answered for SREE as an AFT is, “Which costs a user more, her searching for ambiguities with SREE or her searching for ambiguities totally manually?” To answer this question, we will need to know

- how much time is spent searching for any ambiguity in a totally manual search,
- the amount of time spent in rejecting a false positive, and
- whether knowing the scope of SREE really allows the user to ignore the types of ambiguities in SREE’s scope.

This section uses this information to do an estimated evaluation of the times to use the AFT and to do a manual search to find all ambiguities in the same RS.

The evaluation in this section is not intended to be and is not an empirically sound validation of the effectiveness of SREE as an AFT. Because of the weaknesses reported in Section 7, empirically evaluating the reported version of SREE would be a waste of effort. The sole purpose of the estimation-based evaluation of this section is to determine if it is worth proceeding with the research to design an AFT with the stated goals.

6.1 Time Comparison

When Tjong was doing the research reported herein, she had to do a completely manual examination of each RS that was used in the research and experiments in order to determine which of its Rstats were ambiguous according to the list of ambiguity types that she had built. The New Adelaide Airport RS, with 63 Rstats, required 1.5 hours for its examination, i.e., about 86 seconds per Rstat to determine which of its Rstats were ambiguous. The MCSS RS, with 246 Rstats, required 6 hours, in 2 3-hour sittings, i.e., about 88 seconds per Rstat to make the same determination. The average of these inspection times per Rstat is about 87 seconds.

On the other hand, after she had run SREE on the 22 random Rstats, she spent 3 minutes and 45 seconds or about 10 seconds per Rstat to determine which of the 20 that were marked as potentially ambiguous were truly ambiguous[6]. This determination was essentially instinctive and did not require consulting any lists. That is, without looking at any lists, she could see very quickly what kind of potential ambiguity was present in each potentially ambiguous Rstat and was able to quickly decide if the potential ambiguity was actual. Of course, from her research, she was attuned to finding ambiguities, as would be any experienced ambiguity inspector.

When Tjong ran SREE on the 63 Rstats of the New Adelaide Airport RS [7, pp. A1–A37], SREE marked 42, or 66.66%, of them as potentially ambiguous. When she ran SREE on the 246 Rstats of the MCSS RS [7, pp. A38–A179], SREE marked 201,

[6] The first author had sent these 22 Rstats to others to have them identify the ambiguities they found. These included a PhC in knowledge management, two software engineers, and a marketing executive. None of them found anywhere near the number of ambiguities that the author had found. None of them but the PhC had background in RE and none was a specialist in ambiguity finding.

or 81.70% of them as potentially ambiguous. The weighted average fraction of the RStats that SREE marked as potentially ambiguous was 78.64%. This figure means that on average at about 80% of the Rstats an RS both lie within SREE's scope and are potentially ambiguous. Tjong's running of SREE against the RSs was about a year and a half after she had finished the manual examination of the RSs, enough time that she could not remember the details of what she had done previously, and which sentences were ambiguous.

Once SREE had produced its output from the RSs, Tjong had to examine each marked potentially ambiguous Rstat to determine if it is truly ambiguous. The examination of the 42 marked Rstats of the New Adelaide Airport RS required about 17 minutes, or about 24 seconds per Rstat. The examination of the 201 marked Rstats of the MCSS RS required about 43 minutes, or about 13 seconds per Rstat. The weighted average examination per potentially ambiguous Rstat is about 15 seconds.

So, the choices are:

1. **Completely Manual Inspection:** Manually inspect an entire RS of n Rstats for all types of ambiguities, having to continually consult a list of the types of ambiguities to ensure that none are overlooked, spending about 87 seconds per Rstat, for a total of $87 \times n$ seconds.
2. **Using SREE for the Ambiguities in its Scope and Manual Inspection for the Rest of the Ambiguities:** Run SREE on the RS to obtain a list of potentially ambiguous Rstats, spending about 15 seconds per Rstat to decide which potentially ambiguous Rstat is truly ambiguous. Then manually inspect the entire RS for only the types of ambiguities not in SREE's scope.

   If, as estimated, about 80% of the Rstats of a RS both lie within SREE's scope and are potentially ambiguous, then SREE will mark $.8 \times n$ Rstats as potentially ambiguous and the user will have to spend about 15 seconds to examine each of them, for a total of $12 \times n$ seconds. Then, the user will have to examine the full RS for ambiguities not in SREE's scope. Since at least 80% of the Rstats lie in SREE's scope, a lower bound of the fraction of the Rstats containing potential ambiguities not in SREE's scope is 20%. Of course, there will probably be some Rstats that have a potential ambiguity in SREE's scope and a potential ambiguity not in SREE's scope. So the actual fraction of Rstats containing potential ambiguities not in SREE's scope is more than 20%. We estimate that at worst, about 50% of the Rstats have potential ambiguities not in SREE's scope.

   On the assumption that the user of SREE is very familiar, from lots of practice, with the scope of SREE, we estimate that the examination of the entire RS for potentially ambiguous Rstats outside of the scope of SREE to be about $1.25 \times 87 \times .5 \times n = 54.4 \times n$ seconds, with about 25% more time spent per Rstat than in the normal manual examination of the Rstat to account for having to skip over potential ambiguities in SREE's scope.

Which is larger?

- $87 \times n$ or
- $12 \times n + 54.4 \times n = 66.4 \times n$?

Clearly, the former is larger, and it is about about 23% higher. Thus, the estimated time spent for each Rstat in the mixed-SREE-and-manual-inspection choice is about 75% of the estimated time spent per Rstat in the totally-manual-inspection choice.

As mentioned, this calculated estimated difference depends on the SREE user's having learned to ignore potential ambiguities in SREE's scope to focus her search on finding ambiguities not in SREE's scope. From our experience using other tools and having learned what they do and do not do, we believe that this assumption is reasonable.

The estimated advantage of using SREE in the proper manner over a completely manual search is not very large. In any case, 50% was used as the estimate of the percentage of Rstats that have potential ambiguities not in SREE's scope in order that the comparison be pessimistic on SREE's side. Part of the future work will surely be case studies with careful measurements of the values required for the comparison. Nevertheless, the estimated values are good enough to justify further research to find a better decomposition of the ambiguity finding task that allows an AFT that truly meets SREE's goals.

6.2 Evaluation of Imprecision

The analysis of the SREE runs on the New Adelaide Airport RS shows that there were 58 false positives among the 180 potential ambiguities that SREE found in the RS, for a precision of 68%. The analysis of the SREE runs on the MCSS RS shows that there were 247 false positives among the 723 potential ambiguities that SREE found in the RS, for a precision of 66%. Neither of these precisions is as close to 100% as we had hoped. In retrospect, however, when the AFT does no analysis of what it finds, it is to be expected that a large percentage of what it finds are false positives. Nevertheless, even with this high imprecision, the mixed-SREE-and-manual-inspection time is less than the totally-manual-inspection time. So, it is possible that SREE's imprecision will not be considered burdensome.

6.3 SREE's Weaknesses

SREE's weaknesses that can be observed from our findings are:

- the fact that SREE's scope includes only Pluralnouns and not plural nouns. As explained, Pluralnouns are those tokens recognized as potentially ambiguous by SREE as a result of the Plural corpus, whereas plural nouns is the actual set of nouns that are plural. SREE has 100% recall of Pluralnouns, but not of plural nouns. It may be better to report as potentially plural any word that ends in s and to put into the AIC a list of only irregular plural nouns that do not end in s. A complete enough list of irregular plural nouns that do not end in s is probably smaller than the current list of 11,287 Pluralnouns and is probably easier to make complete enough than a list of all plural nouns. Alternatively, the user is responsible for continuing to add more and more plural nouns to the list of Pluralnouns in the AIC until Pluralnouns has converged on all plural nouns that ever show up in the specifications that the user encounters. This convergence may never really happen due to the inherent and surprising richness of NL. In the end, if too many plural nouns are outside SREE's

scope, then perhaps not even the 11,287 plural nouns in the Plural corpus should be in SREE's scope.
- the fact that the AIC is not complete, as evidenced by the continual, relentless discovery of new indicators to add to the AIC during the research to develop SREE. An outstanding question is whether this AIC will converge for a language or even a domain. We do expect that at some point, the rate of addition of new indicators for other than Pluralnouns will drop considerably, just because we will eventually begin not to find new types of ambiguities. Thus, this work is complementary to all other research work cited in Section 2 that attempts to find systematic ways of detecting or avoiding ambiguities.
- the fact that the SREE's measured precision rate is lower than the goal of not too much less than 100%, particularly because a singular verb that happens to look like an element of Pluralnouns is reported as potentially ambiguous and because a non-subject plural noun matching an element of Pluralnoun is reported as potentially ambiguous (Recall that a plural noun is ambiguous only when it is the subject of a sentence). Despite the low precision, the authors believe that the use of SREE with its 100% recall of potential ambiguities in its scope is better for the user than to have to have to find these potential ambiguities manually in close, error-prone readings of the RS.

7 Conclusion and Future Work

This paper has described the prototyping of SREE, which was designed to be the algorithmic AFT in a decomposition of a process for finding ambiguities in a NL RS into two easily distinguished parts:

1. the algorithmic part that can be done with 100% recall by a tool, the dumb tool, and
2. the nonalgorithmic part that is smaller than manually searching the entire RS and which requires the intelligence of a smart human.

While SREE does not meet all of its its goals, we have learned a lot that can be used in the next attempt to build an AFT according to the suggested decomposition.

The trials showed several weaknesses of SREE and, thus, opportunities for improvement,

- in determining if the list of indicators for the AIC will stabilize for any language or domain, and
- dealing with plural nouns.

The high number of false positives among the potential ambiguities matching indicators in the Plural corpus raises concerns about the usefulness of the chosen method to deal with the plural noun ambiguity. Perhaps, a larger list of plural nouns, including irregulars is needed in the AIC. Alternatively, it may be even better to have SREE's lexical analyzer recognize

1. all words ending in `s`, `es`, `ae`, `aux`, and other common plural noun endings, and
2. all words in a new Plural corpus consisting of as many irregular plural nouns as possible as potential plural nouns.

While this new method will probably find more potentially ambiguous plural nouns than the current method, the new method will probably have higher imprecision than the current method. Ultimately, the issue is which is worse: to have to manually search for plural nouns not currently in SREE's scope or to have more imprecision?

With this method of dealing with plural nouns, there is yet another tradeoff for an AFT designed with the goal of 100% recall as suggested by this paper. If the AFT has a smaller scope then it is easier to reach the goal of 100% recall. However, then the manual search that accompanies the AFT use would have more to do. So it pays to try to find a way to include more potential ambiguities in the scope of any AFT.

Finally, if ever a prototype AFT is developed that passes muster in an informal evaluation like that described in this paper, the AFT must be subjected to a proper, unbiased empirical evaluation, with several users using the AFT applied to several industrial-strength RSs.

Acknowledgements. The authors thank Michael Hartley, Jack, Shindy, and Inn Fang for their comments on all the research reported in this paper. Daniel Berry's work was supported in parts by a Canadian NSERC grant NSERC-RGPIN227055-00 and by a Canadian NSERC–Scotia Bank Industrial Research Chair NSERC-IRCPJ365473-05.

References

1. Berry, D.M., Gacitua, R., Sawyer, P., Tjong, S.F.: The case for dumb requirements engineering tools. In: Regnell, B., Damian, D. (eds.) REFSQ 2011. LNCS, vol. 7195, pp. 211–217. Springer, Heidelberg (2012)
2. van Rossum, W.: The implementation of technologies in intensive care units: Ambiguity, uncertainty and organizational reactions. Technical Report Research Report 97B51, Research Institute SOM (Systems, Organisations and Management), University of Groningen (1997), `http://irs.ub.rug.nl/ppn/165660821`
3. Sussman, S.W., Guinan, P.J.: Antidotes for high complexity and ambiguity in software development. Information and Management 36, 23–35 (1999)
4. Gause, D.C., Weinberg, G.M.: Exploring Requirements: Quality Before Design. Dorset House, New York (1989)
5. Mich, L., Franch, M., Inverardi, P.N.: Market research for requirements analysis using linguistic tools. Requirements Engineering Journal 9(1), 40–56, 9(2), 15 (2004); has full article with inverted names, No. 2 has correction of names and reference to full article in No. 1.
6. Berry, D.M., Kamsties, E.: Ambiguity in requirements specification. In: Leite, J., Doorn, J. (eds.) Perspectives on Requirements Engineering, pp. 7–44. Kluwer, Boston (2004)
7. Tjong, S.F.: Avoiding Ambiguity in Requirements Specifications. PhD thesis, Faculty of Engineering & Computer Science, University of Nottingham, Malaysia Campus, Semenyih, Selangor Darul Ehsan, Malaysia (2008), `https://cs.uwaterloo.ca/~dberry/FTP_SITE/tech.reports/TjongThesis.pdf`
8. Ambriola, V., Gervasi, V.: On the systematic analysis of natural language requirements with CIRCE. Automated Software Engineering 13, 107–167 (2006)
9. Wilson, W.M., Rosenberg, L.H., Hyatt, L.E.: Automated analysis of requirement specifications. In: Proceedings of the Nineteenth International Conference on Software Engineering (ICSE 1997), pp. 161–171 (1997)

10. Fabbrini, F., Fusani, M., Gnesi, S., Lami, G.: Quality evaluation of software requirement specifications. In: Proceedings of the Software and Internet Quality Week 2000 Conference, pp. 1–18 (2000)
11. Fabbrini, F., Fusani, M., Gnesi, S., Lami, G.: The linguistic approach to the natural language requirements, quality: Benefits of the use of an automatic tool. In: Proceedings of the Twenty-Sixth Annual IEEE Computer Society – NASA GSFC Software Engineering Workshop, pp. 97–105 (2001)
12. Kasser, J.: Tiger pro manual. Technical report, University of South Australia (2006), `http://users.chariot.net.au/~g3zcz/TigerPro/tigerPro.pdf`
13. Willis, A., Chantree, F., de Roeck, A.: Automatic identification of nocuous ambiguity. Research on Language and Computation 6, 355–374 (2008)
14. Kiyavitskaya, N., Zeni, N., Mich, L., Berry, D.M.: Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. Requirements Engineering Journal 13, 207–239 (2008)
15. Miller, G.A., Felbaum, C., et al.: WordNet Web Site. Princeton University, Princeton, `http://wordnet.princeton.edu/` (accessed March 12, 2006)
16. Gleich, B., Creighton, O., Kof, L.: Ambiguity detection: Towards a tool explaining ambiguity sources. In: Wieringa, R., Persson, A. (eds.) REFSQ 2010. LNCS, vol. 6182, pp. 218–232. Springer, Heidelberg (2010)
17. Agarwal, R., Boggess, L.: A simple but useful approach to conjunct identification. In: Proceedings of the Thirtieth Annual Meeting of the Association for Computational Linguistics (ACL 1992), pp. 15–21 (1992)
18. Resnik, P.: Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. Journal of Artificial Intelligence Research 11, 95–130 (1999)
19. Goldberg, M.: An unsupervised model for statistically determining coordinate phrase attachment. In: Proceedings of the Thirty-Seventh Annual Meeting of the Association for Computational Linguistics on Computational Linguistics (ACL 1999), pp. 610–614 (1999)
20. Chantree, F., Willis, A., Kilgarriff, A., de Roeck, A.: Detecting dangerous coordination ambiguities using word distribution. In: Recent Advances in Natural Language Processing: Current Issues in Linguistic Theory, vol. 4 (292), pp. 287–296. John Benjamins (2007)
21. Tjong, S.F., Hallam, N., Hartley, M.: Improving the quality of natural language requirements specifications through natural language requirements patterns. In: Proceedings of the Sixth IEEE International Conference on Computer and Information Technology (CIT 2006), pp. 199–206 (2006), `https://cs.uwaterloo.ca/~dberry/FTP_SITE/reprints.journals.conferences/TjongHallamHartley2006A.pdf`
22. Tjong, S.F.: Natural language patterns for requirements specifications. Technical report, Faculty of Engineering & Computer Science, University of Nottingham, Malaysia Campus (2006), `https://cs.uwaterloo.ca/~dberry/FTP_SITE/tech.reports/TjongTR-02_2006.pdf`
23. Tjong, S.F., Hartley, M., Berry, D.M.: Extended disambiguation rules for requirements specifications. In: Proceedings of the Tenth Workshop on Requirements Engineering a.k.a. Workshop em Engenharia de Requisitos (WER 2007), pp. 97–106 (2007), `http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER07/Lwer07-tjongl.pdf`
24. Tjong, S.F., Hallam, N., Hartley, M.: An adaptive parsing technique for prorules grammar. In: Proceedings of the Computer Science and Mathematics Symposium (CSMS) (2006), `https://cs.uwaterloo.ca/~dberry/FTP_SITE/reprints.journals.conferences/TjongHallamHartley2006B.pdf`

25. Collins, M., Duffy, N.: New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In: Proceedings of the Fortieth Annual Meeting of the Association for Computational Linguistics (ACL 2002), pp. 263–270 (2002)
26. Hammerton, J., Osborne, M., Armstrong, S., Daelemans, W.: Introduction to special issue on machine learning approaches to shallow parsing. Journal of Machine Learning Research 2, 551–558 (2002)
27. Bray, I.K.: An Introduction to Requirements Engineering. Addison-Wesley, Harlow (2002)
28. Eng, C.S.: Batch poster system, detailed business requirements. Technical report, EDS MySC (2005)
29. EPRI: Cask loader software, general requirements document draft. Technical Report, Electric Power Research Institute Inc. (1999), `http://www.epri.com/eprisoftware/processguide/docs/srdexdoc.doc`
30. Nelbach, F.: Software requirements document for the data cycle system (DCS). Technical Report, Universities Space Research Association, UCLA (2002), `http://www.astro.ucla.edu/~shuping/SOFIA/Documents/DCS_SRD_Rev1.pdf`
31. Moeser, R., Perley, P.: Expanded very large array (EVLA) operations interface software requirements. Technical Report EVLA-SW-003, National Radio Astronomy Observatory (2003), `http://www.aoc.nrao.edu/evla/techdocs/computer/workdocs/array-sw-rqmts.pdf`
32. Dubois, R.: Large area telescope (LAT) science analysis software specification. Technical Report GE-0000X-DO, SLAC National Accelerator Laboratory (2000), `http://www.last.slac.stanford.edu/IntegrationTest/DataHandling/docs/LA%T-SS-00020-6.pdf`
33. George, S.: PESA high-level trigger selection software requirements. Technical Report, Centre for Particle Physics at Royal Holloway University (2001), `http://www.pp.rhul.ac.uk/atlas/newsw/requirements/1.0.2/`
34. Stevenson, M., Hartley, M., Iacovou, H., Tan, A., Phan, L.: Software requirements specification for sort algorithm demonstration program. Technical report, SDPM (2005)