Requirements for Tools for Ambiguity Identification and Measurement in Natural Language Requirements Specifications

Nadzeya Kiyavitskaya[1] (nadzeya@dit.unitn.it),
Nicola Zeni[1] (nicola.zeni@unitn.it),
Luisa Mich[2] (luisa.mich@unitn.it), and
Daniel M. Berry[3] (dberry@uwaterloo.ca)

[1] Department of Information and Communication Technologies, University of Trento, Italy
[2] Department of Computer and Management Sciences, University of Trento, Italy
[3] Cheriton School of Computer Science, University of Waterloo, Canada

Abstract. This paper proposes a two-step approach to identifying ambiguities in natural language (NL) requirements specifications (RSs). In the first step, a tool would apply a set of ambiguity measures to a RS in order to identify potentially ambiguous sentences in the RS. In the second step, another tool would show what specifically is potentially ambiguous about each potentially ambiguous sentence. The final decision of ambiguity remains with the human users of the tools. The paper describes several requirements-identification experiments with several small NL RSs using four prototypes of the first tool based on linguistic instruments and resources of different complexity and a manual mock-up of the second tool.

1 Introduction

Ambiguity is an intrinsic phenomenon of natural language. It means the capability of being understood in two or more possible senses or ways. Identification of ambiguous words and phrases is a crucial aspect in text-processing applications and many other areas concerned with human communication. The main focus of the present work is the problem of ambiguity identification in natural language documents, in particular with natural language (NL) requirements specifications (RSs) for computer-based systems (CBSs).

The main goals for any tool for identifying and measuring ambiguities in NL RSs are: (1) to identify which sentences in a NL RS are ambiguous and, (2) for each ambiguous sentence, to help the user to understand why it is ambiguous, so that he can remove the ambiguity from the sentence, and thus improve the NL RS.

There have been several attempts and proposals to apply linguistic tools to the requirements engineering (RE) problem of identifying and eliminating ambiguity in RSs for CBSs [1, 2, 3, 4, 5]. Despite the hopes raised by the success of such tools [e.g., 6, 7, 8, 9], in other domains, e.g., in message understanding as evidenced by the annual Message Understanding Competition [10], these RE attempts have not been complete.

This paper proposes a two-step approach to identifying ambiguities in NL RSs. In the first step, one tool, T1, would be used to apply of set of ambiguity measures to a

RS in order to identify potentially ambiguous sentences in the RS. In the second step, another tool, T2, would show what specifically is potentially ambiguous about each sentence in the RS. Since the final decision of whether a sentence is ambiguous rests with the human users of the tooks, any sentence that either tool tags as potentially ambiguous is really only *potentially* ambiguous.

This paper describes work to determine requirements for T1 and T2. In this work, T1 was prototyped by shell scripts that invoke commands offered by a general-purpose NL processing (NLP) system in order to calculate a set of ambiguity measures that can be applied to the sentences of any NL RS, and for that matter, of any NL document. Actually, T1 went through four prototyping iterations, the first based on one NLP system and the remaining three based on various linguistic instruments and resources. T2 was prototyped by having the human authors of this paper search for instances of a collection of ambiguities identified in the literature as appearing in NL RSs.

Therefore, Section 2 of this paper reviews the ambiguity problem. Section 3 reviews the main work concerning ambiguity identification both for general text and for RS text. Section 4 describes the four iterations of T1 and experiments involving their applications to small NL RSs. Section 5 describes T2 and an experiment of its application to one of the small NL RSs used in the work described in Section 4. Conclusions are drawn in Section 6.

2 Overview of Ambiguity

Ambiguity is a pervasive phenomenon in human languages, and is fundamentally a property of linguistic expressions. There are two basic definitions of “ambiguity”:

1. the capability of being understood in two or more possible senses or ways;
2. uncertainty [12].

Uncertainty means lack of sureness about something, often because of gaps in the writer’s or reader’s or both’s background knowledge. The issue of uncertainty is not considered in this paper; thus, the paper uses the first definition of “ambiguity”. This paper uses the coined term “uniguity” to mean the lack of ambiguity.

A word, phrase, sentence, or other message is called *ambiguous* if it can be reasonably interpreted in more than one way. It is difficult to find a word that does not have at least two possible meanings, and an isolated sentence, separated from its context, is often ambiguous.

The traditional types of ambiguity include lexical, syntactic, semantic, and pragmatic ambiguity. To this list we add two additional types, software-engineering, and language-error ambiguity [13]. Each of most of these types has subtypes, and the elements of an occasional pair of subtypes share a parent type. For more details on these types of ambiguity, see the survey by Berry and Kamsties [13].

Another way to view the two tools is that T1 is focused on measuring lexical and syntactic ambiguities, and T2 is focused on identifying specific instances of pragmatic, software-engineering, and language-error ambiguities. Handling semantic ambiguity requires language understanding, which is beyond the scope of these tools [14]. However, some specific instances of semantic ambiguity can be caught or approximated by lexical or syntactic means, and these instances could show up among the ambiguities measured or identified by T1 or T2.

Ambiguity gives NL its flexibility and usability. Consequently, ambiguity cannot be entirely eliminated from any NL. Even in a situation in which some ambiguity can or should be eliminated, it cannot be completely eliminated, just because a reader is often not aware of ambiguities in what she is reading. Arising from this lack of awareness is the insidious phenomenon of subconscious disambiguation [15]. The reader, unaware of any other reading of a sentence, uses the first meaning she understands of the sentence, which may not be the meaning the writer intended. Moreover, the writer was equally unaware of the possible other readings of what he wrote.

While ambiguity may be useful in some uses of NL, ambiguity in an early or late NL RS can cause numerous problems. An analyst's subconscious disambiguation of an early NL RS can lead to the wrong CBS's being specified in the resulting more formal RS. An implementer's subconscious disambiguation of a late NL RS can lead to the wrong implementation's being built. Finally, a tester's subconscious disambiguation of an early or late NL RS can lead to the wrong test cases' being applied, the wrong answers' being accepted as correct, or the correct answers' being rejected as wrong.

3 Related Work in Ambiguity, Disambiguation, Tools, and Prevention

Much work has been done in the field of ambiguity, and a number of linguistic theories have been developed [e.g., 16, 17, 18, 19, 20]. Resolving ambiguities is a requirement for many NL understanding applications. Indeed, for many in the NL understanding area, disambiguation *is* NL understanding [e.g., 17]. Note that disambiguation requires at least implicit ambiguity identification.

Ide and Véronis [21] report on the history of the word-sense disambiguation (WSD) field until 1997. Yarowsky *et al* improved WSD by incorporating statistical techniques [22]. At the end of 1997, an international organization, SENSEVAL [23] was formed to evaluate the quality of WSD systems. The core mission of SENSEVAL is to organize and run tests of the strengths and weaknesses of implemented WSD systems against different words, different aspects of language, and different languages. SENSEVAL has spurred the development of practical strategies for analyzing NL text by providing a test bed to be used by any candidate tool.

One of the first machine translation programs, developed by Harper [24, 25], estimated the degree of polysemy of any text it was trying to translate. For example, in the process of translating Russian language physics articles to English, Harper's program determined that about 30% of the words in article and about 43% of the words in another article are polysemous. Values such as these could be used to calibrate acceptable levels of polysemy for articles in articles of any domain. Harper estimated the degree of polysemy also in dictionaries.

Some have considered the application of ambiguity identification in RE to help improve the quality of NL requirements specifications. The tools developed so far use lexical and syntactical analysis to identify ambiguities and to measure the degree of ambiguity in a NL RS. Some of these tools try to measure also vagueness, subjectivity, optionality, and weakness of the RS. One class of tools are those developed specifically for NL RS ambiguity identification and measurement (NLRSAI&M). These include QuARS [3], ARM [1], KANT [26], and Chantree's tool [27]. Another class of tools are those developed for general linguistics purposes, but are applied to NLRSAI&M. These include LOLITA [7, 8, 2].

For instance, the linguistic tool QuARS (Quality Analyzer of Requirement Specification) [3] has been applied to evaluate industrial RSs. The tool is based on a quality model (QM) for NL RSs. The QM specifies lexical, syntactic, structural, and semantic defects that appear in NL RSs. Among these defects are ambiguities. QuARS uses linguistic algorithms to implement the detection of potential instances of those defects. QuARS highlights each potential instance it finds in a RS and leaves it to the the user to decide whether or not to modify the RS. Basically, QuARS's semi-automatic approach is to use a set of lexical, syntactic, and structural indicators of likely defects including ambiguities. For example, the lexical indicators consist of a list of keywords, each of which indicates a common defect. A user can add domain-specific defect indicators to the set for which QuARS searches. Recently, the QM has been extended with indicators of kinds of defects, including ambiguities, not detected by the current version of QuARS [14]. The intention is that detection of these new indicators be added to QuARS.

NASA (National Aeronautics and Space Administration) has developed a similar tool, ARM (Automated Requirement Measurement) [1], for automated analysis of the quality of their RSs. ARM employs an approach similar to that of QuARS. Also ARM is based on a QM, and also ARM highlights potential defects, such as weak and ambiguous phrases, in SRSs. As with QuARS, an ARM user can add domain-specific defect indicators to the set for which ARM searches.

Some techniques try to minimize the number of ambiguities rather than to resolve ambiguities. For example, the KANT machine translation system [26] introduces some restrictions on the input NL text. These restrictions include a constrained lexicon, a constrained grammar, constrained noun–noun compounding, and a domain model to constrain the semantics. KANT allows also interactive disambiguation of text. Its online authoring system is able to indicate potential lexical and structural ambiguities in any sentence. If the author agrees with KANT's assessment of ambiguity, he can rewrite the sentence.

Chantree [27] considers ambiguity detection in NLG. He proposes an ambiguity notification tool similar to KANT. Chantree's system identifies ambiguity in text being generated and highlights them to the user, who can choose either to accept or to change the text. Chantree's innovation over KANT is to add measures of an ambiguity's seriousness level and of a user's tolerance of ambiguity. An ambiguity's *seriousness* is a measure of the ambiguity's criticality. An initial assessment of each ambiguity's seriousness is provided by domain experts working with knowledge of the ambiguity's context. A user's *tolerance of ambiguity* is the highest ambiguity seriousness below which he tolerates an ambiguity. When built, the tool would adjust these measures during interaction with a user, and the user would be able to reset either measure at any point.

Mich and Garigliano [2] investigated the use of indices of ambiguity in NL text to evaluate NL RSs. The value of the ambiguity index for a word is computed as a weighted function of (1) the number of semantic meanings of and (2) the number of syntactic roles for the word. The weights depend on the frequencies of the occurrences of the different meanings and roles in the language of the RS. The ambiguity index of a sentence is defined as a weighted function of the ambiguity indices of the sentence's words. Mich and Garigliano use Garigliano's general-purpose NL Processing (NLP) system LOLITA [7, 8] to do the ambiguity index calculations. The ambiguity identification and measurement tool described in Section 4 is based on this work by Mich and Garigliano.

Note that almost every tool requires some restrictions on the input NL text, even if it is only in the vocabulary used. Moreover, almost every tool-based approach assumes semi-automatic use of the approach's tool, in which the tool asks the user for help or presents to the user choices that must be made.

Generally, no ambiguity identification tool can be perfect; it will fail to find some, i.e., it will not have total *recall,* and it will find what are not really ambiguities, i.e., it will not have total *precision*; therefore, a tool can at best show only potential ambiguities. Once shown a potential ambiguity, the user can determine if the potential ambiguity is real, and if so, she can rewrite the offending text. Of course, not all ambiguities can be easily identified. Finding some of them requires deep linguistic analysis.

Others have considered approaches to help RS writers to write less ambiguously, e.g., with patterns based on a metamodel of RS statements [28, 29, 30] or with a restricted language in an unambiguous sublanguage of one's NL [31, 32, 33].

Kamsties, Berry, and Paech [28] suggest using a metamodel of RS sentences as patterns to allow identification of ambiguities in NL RSs. The metamodel needs to be adapted to the domain of the RS to be analyzed. Kamsties, Berry, and Paech describe pattern-driven inspection techniques, namely checklists and scenario-based reading, whose effectiveness in detecting ambiguities in NL RSs has been empirically validated.

Fuchs and Schwitter describe Attempto Controlled English (ACE), a sublanguage of English consisting of only uniguous sentences. Each of its sentences can be translated by the ACE translator into a sentence in first-order logic. [31, 32].

4 Requirements for T1

The purpose of T1 is to apply of set of ambiguity measures to a RS in order to identify potentially ambiguous sentences in the RS. This section describes the work we did to identify requirements for T1 by building prototypes for T1 and applying the prototypes to NL RSs. T1 went through four prototyping iterations: the first, $T1_1$, based on one NLP system and the remaining three, $T1_2$, $T1_3$, and $T1_4$, based on various linguistic instruments and resources.

Mich and Garigliano [2, 4] constructed an unnamed tool, called $T1_1$ in this paper, as a script invoking commands offered by Garigliano's previously constructed general-purpose NLP system called LOLITA. The script implemented a collection of specific ambiguity measures described in Section 4.2. The goal of building $T1_1$ was to demonstrate that an existing general-purpose NLP system could be used as a platform on which to build a tool for NLRSAI&M. $T1_1$ largely met its goal, but

1. $T1_1$ is very expensive to run because of the high overhead of running its underlying platform LOLITA, which is doing many more things than are needed for NLRSAI&M;
2. LOLITA has gone commercial, and we cannot afford a license to use it and to redistribute it along with $T1_1$; and
3. LOLITA's dictionary is weak because the focus of its builders was to make a general purpose NLP system and not a complete dictionary.

$T1_1$'s effectiveness was demonstrated in experiments applying $T1_1$ to several RSs.

Thus, the goal became to build a new version of T1, called $T1'$.

1. $T1'$ would be lighter weight than T1, as it computes only what is necessary for NLRSAI&M.
2. $T1'$ would be based on publically accessible resources, i.e., data and software, combined with easily written scripts.
3. $T1'$ would use a publically accessible dictionary built with the goal of being a complete dictionary.

Because we did not understand fully the requirements for $T1'$, we decided to prototype $T1'$ by manually invoking available and quick-and-dirty software to simulate whatever we thought should be $T1'$'s behavior. We planned to do a sequence of these manually simulated prototypes to answer a series of questions about $T1'$'s requirements.

For the parse-tree-based functions, we determined fairly quickly that we could use any of the publically accessible parse-tree generators in place of LOLITA's parser. These parse-tree generators include that built by Sleator and Temperly (S&T) at Carnegie Mellon University [34] and TreeTagger, built by the Institute for Computational Linguistics at the University of Stuttgart [35, 36]. A quick inspection of these showed that each performed at least as well as LOLITA's parse-tree generator. Therefore, the focus of the prototyping experiments was on the lexical-ambiguity functions based on a dictionary.

The goal of building the first manually simulated prototype, $T1_2$, was to determine which of three publically accessible online lexical resources is the best for the purpose of efficiently and effectively calculating the lexical-ambiguity measures of $T1'$. An experiment applying $T1_2$ to sets of menu item names allowed choosing one particular lexical resource, the thesaurus provided by WordNet [37].

The goal of building the second manually simulated prototype, $T1_3$, was to determine which of two possible auxiliary functions needed to calculate the lexical ambiguity of a sentence is the best for the purpose of NLRSAI&M. An experiment applying $T1_3$ to one RS to which T1 was applied allows choosing one particular auxiliary function.

We built $T1_4$ from $T1_3$ by freezing the auxiliary function parameter to the chosen function and from $T1_2$ by freezing the lexical resource parameter to the chosen resource. In addition, we tried one particular user interface (UI) in $T1_4$ to see if it is helpful. $T1_4$'s effectiveness in measuring the lexical ambiguity of sentences was tested in experiments applying $T1_4$ to three NL RSs, including the one to which we applied $T1_1$.

4.1 Design and Construction of $\mathrm{T1}_1$ Based on LOLITA

In the late 1990s, Luisa Mich and Roberto Garigliano developed $\mathrm{T1}_1$, a LOLITA-based tool for calculating lexical, syntactic, and semantic ambiguities of words and sentences [2, 4]. They constructed $\mathrm{T1}_1$ as a module by using commands of the LOLITA NLP system, which is a general-purpose, domain-independent NLP system designed for production use [38]. All the morphological, grammatical, semantic, pragmatic, etc. data used by LOLITA are stored in a large semantic net that serves as LOLITA's knowledge base. The version of LOLITA used to support the ambiguity calculating $\mathrm{T1}_1$ has a net of about 150,000 nodes connected in hierarchies. LOLITA accepts input in English, but it has data also for Spanish, Chinese, and Italian. When LOLITA is presented with a NL document as input, LOLITA analyzes the document morphologically, syntactically, and then semantically. The semantic analysis yields a graph that is added to the semantic net. LOLITA then analyzes these newly attached parts of the semantic net pragmatically; this pragmatic analysis consists of checking for consistency with the rest of the semantic net and adding new information to the semantic net.

Among the information LOLITA determines for each parse tree t of a sentence S is the penalty of t as the intended parse tree of S. The penalty of a parse tree t of s is LOLITA's statement of how much effort it spent building t. This penalty is an attempt to model the likelihood for t to be the parse tree intended by the person who said or wrote S. That is, the higher the penalty of t, the less likely that t is the parse tree the author of S intended. LOLITA offers to the user the tp command that can rank the parse trees of a sentence S according to each tree's penalty and can output each tree with its penalty attached to it.

In LOLITA, the names and meanings of the specific penalty values, from highest to lowest, and in the format "*name* :*meaning*" are:

4: a tree with a penalty value of greater than or equal to 1000 has major structural problems, such as an apparent or real missing or repeated part of speech, e.g. zero or two verbs as in He verbs nouns and nouns verbs.[1],

3: a tree with a penalty value less than 1000 but greater than 100 has a major feature clash, such as an apparent or real dative or infinitive use of inappropriate verbs, e.g., I sent the user data. or I lent my son my maid.,

2: a tree with a penalty value less than 100 but greater than 30 has a minor feature clash, such as a wrong concordance, e.g., That is so twentieth century.,

1: a tree with a penalty value less than or equal to 30 but greater than 0 has at most some less common but nevertheless correct constructs, e.g. a noun used as an apposition to another noun, which is less common than an adjective used as apposition to a noun, and

0: a tree with a penalty value less than or equal to 0 has no problems whatsoever.

[1] From now on, any example text is given in a sansserif typeface in order to reserve quotation marks for surrounding a quotation, the meaning of an example, and a non-example word used as itself. Morevoer, when an example ends with punctuation, that punctuation is given in the sansserif typeface and should be distinguished from possibly following punctuation, given in the serif typeface, that belongs to the surrounding sentence.

Thus, it is desirable to find at least one parse tree for S with its penalty value being less than or equal to 30. The scale of penalty values is exponential. Therefore, we have gotten used to calling each penalty value by a number proportional to the logarithm of the lower bound of the range the value is in, namely the item numbers of the descriptions of the ranges given just above. Also, we collapse the range called "0" into the range called "1".

4.2 Ambiguity Measures Computed by LOLITA and T1$_1$

T1$_1$ is capable of calculating several measures on the words and on the sentences of the input NL document:

1. lexical ambiguity of a word W:

$$\alpha(W) = \text{the number of meanings of } W \text{ in LOLITA's semantic net,} \tag{1}$$

2. frequency-weighted lexical ambiguity of a word W:

$$\alpha^*(W) = \sum_{i=1}^{\alpha(W)} \log_2 F(M_i(W)) \tag{2}$$

   where $M_i(W)$ is the ith meaning of W in LOLITA's semantic net, and $F(m)$ is the frequency among meanings of W of the meaning m of W in LOLITA's semantic net,
3. syntactic ambiguity of a word W:

$$\begin{array}{r}\beta(W) = \text{the number of syntactic roles, a.k.a} \\ \text{parts of speech, of } W \text{ in LOLITA's semantic net,}\end{array} \tag{3}$$

   Observe that for each word W, $\beta(W) \leq \alpha(W)$.
4. frequency-weighted syntactic ambiguity of a word W:

$$\beta^*(W) = \sum_{i=1}^{\beta(W)} \log_2 F(R_i(W)) \tag{4}$$

   where $R_i(W)$ is the ith syntactic role of W in LOLITA's semantic net, and $F(r)$ is the frequency among syntactic roles of W of the syntactic role r of W in LOLITA's semantic net,
5. lexical[2] ambiguity of a sentence S:

$$\gamma(S) = \sum_{i=1}^{\#(S)} \alpha(S_i) \tag{5}$$

   where $\#(S)$ is the number of words in S and S_i is the ith word of S,

[2] The literature calls this function "semantic ambiguity" for two reasons: (1) some approximate semantic ambiguity with lexical ambiguity, and (2) the word "lexical" applies to individual words and is somewhat meaningless when applied to a whole sentence.

6. frequency-weighted lexical ambiguity of a sentence S:

$$\gamma^*(S) = \sum_{i=1}^{\#(S)} \alpha^*(S_i) \tag{6}$$

where $\#(S)$ is the number of words in S and S_i is the ith word of S,

7. syntactic ambiguity of a sentence S:

$$\delta(S) = \text{the number of parse trees of } S \text{ reported by LOLITA's parser,} \tag{7}$$

8. penalty of a parse tree t of a sentence S:

$$\pi(t, S) = \text{the name of LOLITA's penalty range of } t \text{ as a parse tree of } S, \tag{8}$$

9. minimum penalty of a sentence S:

$$\pi(S) = \text{Min}_{i=1}^{\delta(S)} \pi(t_i, S) \tag{9}$$

where t_i is the ith parse tree among the $\delta(S)$ parse trees of S,

10. penalty-weighted syntactic ambiguity of a sentence S:

$$\delta^*(S) = \delta(S) \times \pi(S) \tag{10}$$

11. lexical ambiguity of a word w in a sentence S according to a parse tree t of S:

$$\begin{aligned}\alpha^{t,S}(w) = {} & \text{the number of meanings of } w \text{ in LOLITA's} \\ & \text{semantic net that have the syntactic role } r,\end{aligned} \tag{11}$$

where r is the syntactic role of w in t, which is a parse tree of S, and

12. syntax-weighted lexical ambiguity of a word w in a sentence S according to the parse trees of S:

$$\alpha^S(w) = \frac{\sum_{i=1}^{\delta(S)} \alpha^{t_i,S}(w)}{\delta(S)} \tag{12}$$

where t_i is the ith parse tree among the $\delta(S)$ parse trees of S.

For example, LOLITA's lc command shows all the meanings associated in LOLITA's semantic net with its input word W. It thus exhibits $\alpha(W)$ and $\beta(W)$. For the word bank, lc reports 13 different meanings, of which 7 are as nouns and 6 are as verbs. Thus α(bank) = 13 and β(bank) = 2. The large number of meanings is due to the size of the semantic net of LOLITA. Each node in the net represents a single meaning, which may or may not be relevant to the current context. For the example bank, there are meanings as a financial institution, as a financial institution's building, as a river's edge, etc. If the context is fixed by applications that concern businesses, the meaning of bank as a river bank can usually be ignored.[3]

[3] However, one could have a sentence talking about a branch bank of Royal Bank that is close to the bank of a branch of the Credit River. (Believe it or not, there is a river named "Credit River" not too far from a main branch bank of the Royal Bank of Canada on Highway 401 between Waterloo and Toronto in Ontario, Canada.)

LOLITA's pasbr command shows all the parse trees LOLITA finds for its input sentence S in a format that allows determining the various lexical ambiguity values of S. In this format, for each parse tree t for S, for each word w appearing in S, the syntactic role of w in t and the lexical ambiguity of w according to t are given. Since the lexical ambiguity of w according to t counts only the meanings of w that are relevant to w's syntactic role in t, this lexical ambiguity should be less than or equal to the simple lexical ambiguity of w given by lc.

Also LOLITA's tp command shows all the parse trees LOLITA finds for its input sentence S, but showing with each parse tree its penalty. This output combined with that of pasbr allows computing all the penalty-weighted ambiguity measures.

T1_1 computes all of the listed measures except those dependent on the use frequency of words in normal NL text. That is, for any word W, any sentence S, any parse tree t in S, and any word w in S, T1_1 computes the functions: (1) $\alpha(W)$, (3) $\beta(W)$, (5) $\gamma(S)$, (7) $\delta(S)$, (8) $\pi(t,S)$, (9) $\pi(S)$ (10) $\delta^*(S)$, (11) $\alpha^{t,S}(w)$, and (12) $\alpha^S(w)$.

4.3 Experiment with T1_1

Mich [4] describes using the LOLITA-based tool on the ABC Video Problem statement (ABCVPS), a simple RS for a video tape rental system for the ABC Video company [39]. Figure 1 shows the text of the ABCVPS with each sentence numbered. Table 1

1. Customers select at least one video for rental.
2. The maximal number of tapes that a customer can have outstanding on rental is 20.
3. The customer's account number is entered to retrieve customer data and create an order.
4. Each customer gets an id card from ABC for identification purposes.
5. This id card has a bar code that can be read with the bar code reader.
6. Bar code Ids for each tape are entered and video information from inventory is displayed.
7. The video inventory file is updated.
8. When all tape Ids are entered, the system computes the total bill.
9. Money is collected and the amount is entered into the system.
10. Change is computed and displayed.
11. The rental transaction is created, printed and stored.
12. The customer signs the rental form, takes the tape(s) and leaves.
13. To return a tape, the video bar code ID is entered into the system.
14. The rental transaction is displayed and the tape is marked with the date of return.
15. If past-due amounts are owed they can be paid at this time; or the clerk can select an option which updates the rental with the return date and calculates past-due fees.
16. Any outstanding video rentals are displayed with the amount due on each tape and the total amount due.
17. Any past-due amount must be paid before new tapes can be rented.

Fig. 1. ABC Video Problem Statement with Sentences Numbered

shows the syntactic ambiguity measures $\delta(S)$, $\delta^*(S)$, and $\pi(S)$ calculated from the

Penalty-Weighted Ambiguity $= \delta^*(S)$	Sentence Subjected to LOLITA-Based Tool $= S$	Number of Trees $= \delta(S)$	Mininum Penalty Range $(\pi(S))$
40	Customers select at least one video for rental.	10	$\geq$1000 (4)
10	Customers select at least one video to rent.	10	$\leq$30 (1)
8	The maximal number of tapes that a customer can have outstanding on rental is 20.	2	$\geq$1000 (4)
6	The maximal number of tapes that a customer can have on rental is 20.	2	$>$100&$<$1000 (3)
14	The customer's *account* number is entered to retrieve customer data and create an order.	14	$\leq$30 (1)
4	The account number of the customer is entered to retrieve customer data and create an order.	4	$\leq$30 (1)
4	Each customer gets an id *card* from ABC for identification purposes.	1	$\geq$1000 (4)
4	This id *card* has a bar code that can be *read* with the bar code reader.	6	$\leq$30 (1)
?	Bar code Ids for each tape are entered and video information from inventory is displayed.	2	?
1	The video inventory file is updated.	1	$\leq$30 (1)
8	When all tape Ids are entered, the system computes the total bill.	2	$\geq$1000 (4)
2	Money is collected and the amount is entered into the system.	2	$\leq$30 (1)
1	Change is computed and displayed.	1	$\leq$30 (1)
2	The rental transaction is created, printed and stored.	2	$\leq$30 (1)
1	The customer signs the rental *form*, *takes* the tape(s) and leaves.	1	$\leq$30 (1)
?	To return a tape, the video bar code ID is entered into the system.	2	?
8	The rental transaction is displayed and the tape is *marked* with the date of *return*.	2	$\geq$1000 (4)
48	If past-due amounts are owed they can be paid at this time; or the clerk can select an option which updates the rental with the return date and calculates past-due fees.	12	$\geq$1000 (4)
32	Any outstanding video rentals are displayed with the amount due on each tape and the total amount due.	8	$\geq$1000 (4)
72	Any past-due amount must be paid before new tapes can be rented.	18	$\geq$1000 (4)

Table 1. LOLITA-Generated Data for ABC Video Problem Statement Lines and Variants

outputs of the application of various commands of LOLITA to each of the sentences of the ABCVPS.

Each row shows the data for one sentence. The second column shows the sentence. The first column gives the penalty-weighted ambiguity, the δ^*, of the sentence. This value, which is the product of the values in the third and fourth columns, is given in the first column to allow quick determination of which sentences are regarded as most ambiguous. The third column gives the number of parse trees, the δ, for the sentence and the fourth column gives the range of the lowest penalty calculated for these parse trees, followed by the numerical name of the range, i.e., π of the sentence.

A word in a sentence S is italicized if the word has the highest syntax-weighted lexical ambiguity among all the words in all the parse trees of S. When a slight variation of an original sentence in the ABCVPS is given in the row underneath that of the original sentence, the variation is one that has fewer parse trees or a lower minimum penalty among its parse trees. The text that was replaced in the variation was regarded by LOLITA as making the sentence particularly ambiguous. In rows in which a "?" is given as the minimum penalty value, the tp command timed out when processing the row's sentence.

Notice the range of values in the first column, that of the penalty-weighted ambiguity, or δ^*, of sentences. The values range from a low of 1 to a high of 72. From our experience, it seems right to classify a value of less than or equal to 5 as signifying "little or no ambiguity", a value of greater than or equal to 20 as signifying "highly ambiguous", and a value of greater than 5 and less than 20 as signifying "somewhat ambiguous".

There are a number of specific observations about the data in this table.

– Each of about half of the sentences has only 1 or 2 parse trees, well within what is considered little or no ambiguity.
– The analysis for each row was obtained by analyzing the row's sentence in the context of the complete list of sentences of the ABCVPS. In the case of a row that is a variation of one of the first three sentences, the context contains the original variation for the other of the first three sentences. The results would be very different if each sentence were analyzed separately.

Data obtained from many uses of LOLITA in many domains [8, 7] show that in general,

– each of 20% of the sentences has only one parse tree,
– each of 25% of the sentences has between 2 and 9 parse trees,
– each of 47% of the sentences has 10 or more parse trees,
– each of 3% of the sentences has no parse tree, i.e., it is not really a sentence, and
– each of 5% of the sentences takes so long to parse that LOLITA times out.

Thus, the ABCVPS is less ambiguous than the typical NL document.

The questions remain: "How ambiguous is a sentence that LOLITA says has more than one parse tree?" "When LOLITA finds more than one parse tree for a sentence, is the sentence really ambiguous?" The answer, despite the feelings of a human reader, is "Yes!" LOLITA finds more parse trees than any human will find because LOLITA takes into account *all* possible interpretations that are syntactically correct while a human totally ignores all the parse trees that she belives are clearly not intended by the sentences's author. The human practices what is known as subconscious disambiguation [15].

The experiment shows that $\mathrm{T1}_1$ is effective at finding syntactic ambiguities in NL RSs. However, the use of $\mathrm{T1}_1$ requires a running LOLITA. LOLITA computes a lot of information that is not needed for NLRSAI&M and $\mathrm{T1}_1$. Thus, $\mathrm{T1}_1$ suffers from the overhead of LOLITA's calculations. Moreover, as mentioned, LOLITA is now commercial. Therefore, a user of $\mathrm{T1}_1$ has to have a license for LOLITA.

The experiment with $\mathrm{T1}_1$ focused on the ambiguity measures based on the parse tree, i.e., $\delta(S)$, $\delta^*(S)$, and $\pi(S)$. It ignored the ambiguity functions that depend on the number of senses per word, which in turn depends on a dictionary. From experiments not presented in this paper, it became apparent that LOLITA's use of its own semantic network was limiting the effectiveness of LOLITA's calculations of measures based on the number of word senses. Even though LOLITA's semantic net was at that time larger than Wordnet's network and even though LOLITA's semantic network was built to support a general-purpose NLP system, LOLITA's semantic network contained fewer senses per word than any existing dictionary, because LOLITA knew word senses related to specialized meaning for only a limited numbers of domains. So, we began to hunt for a better dictionary to use, from a source focused on making its dictionary complete. Thus the two goals of the subsequent work was to find a less expensive basis NLP system and a better dictionary.

4.4 Prototyping of $\mathrm{T1}_2$

The purpose of constructing $\mathrm{T1}_2$ was to determine which of three publically accessible online dictionary resources is the best for the purpose of efficiently and effectively calculating the lexical-amgiguity measures, which depend on a dictionary.

Dictionaries have been traditionally used to identify lexical ambiguity, because people usually refer to meanings reported in a dictionary when talking about the senses of a word. A dictionary strives to describe the meanings of all, or at least as many as possible, senses of each word in it, and it can be used by a person also to determine the right sense for any occurrence of a word the person has seen or heard. People use also thesauri to help identify lexical ambiguity. A thesaurus gives for each word in it a list of senses and for each sense, a list of synonyms and antonyms.

The experiment with $\mathrm{T1}_2$ focuses on the simplest task, i.e. identification of lexical ambiguity without the use of any syntactic information. It uses publically accessible, machine-readable lexical resources in order to identify lexical ambiguity based on the numbers of senses of words. Two of these resources are dictionaries and the third is a thesaurus:

– WordReference [40] is based on the Collins English dictionary, which covers a wide range of fields. Among the chosen dictionaries, it has the largest average number of senses per word.
– WordNet [37] is probably the most popular of the resources available to researchers in computational linguistics, text analysis, and related areas. The main feature of

WordNet is that it has a semantic network that allows the senses of words to be semantically related to each other.

– Babylon's English dictionary [41] is a huge English language resource, comprising general, encyclopedic, slang, and informal terms. It covers a wide range of professional fields.

We chose these particular resources because

1. they are available at no cost,
2. they are accessible on the Web, allowing the user to quickly browse their knowledge bases,
3. they have friendly user interfaces that are integrated with familiar text editors, familiar ways to interact, and familiar ways to represent output,
4. they have functionalities that suit our needs; each resource provides for each word, a list of its senses, and for each sense, its syntactic role, i.e., its part of speech, and
5. they are heterogeneous resources, in the sense that each of WordReference and Babylon is a dictionary, and WordNet is commonly considered a thesaurus.

Even though one lexical resource is officially a thesaurus, for simplification of the discussion, this paper calls each lexical resource a "dictionary". "Dictionary" is shorter than "lexical resource", and in fact, the one thesaurus is being used as a dictionary.

Table 2 reports the total number of word senses in each dictionary, as of December 2002, when the first of these experiments were done. The resources are difficult to

Resource	WordReference	WordNet	Babylon
word sense equivalent	headword	string	definition
number of word sense equivalents in resource	180,000	144,309	138,814

Table 2. Dimensions of the Dictionaries

compare, because "word sense" is defined differently for each resource. The table shows for each resource what was used as its equivalent of "word sense".

4.5 Experiment with $\mathrm{T1}_2$

The goal of the experiment with $\mathrm{T1}_1$ was to determine which lexical resource to use to measure the ambiguity of single words. The artifact analyzed for ambiguity of its words was a program's menus. Mich and Garigliano [2] suggested that a program's menus would be a good test of a tool for RE because proper RE of the user interface of a menu-driven program includes analyzing the program's menus. We chose as the first experiment's artifact the menus of a popular CASE tool, the May 2002 Rational Solutions version of Rational Rose [42].

We analyzed Rose's first two menus, File and Edit, which exist in many other applications. For each word w which is a menu item of either of these menus, we counted the number of senses and the number of parts of speech there are for w in each of the dictionary. Recall that we are using the number of senses of a word as the word's lexical ambiguity.

As can be seen from the graph in Figure 2, the word cut is the most polysemous

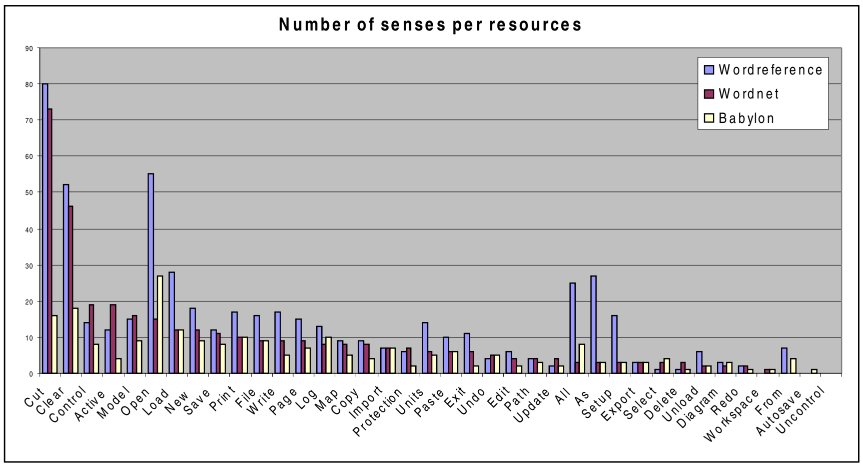


Fig. 2. Number of Word Senses in the Three Dictionaries per Menu Item

in each of WordNet and WordReference, but the word open is the most polysemous in Babylon.

To evaluate the ambiguity of the menu item words independently of dictionaries, we calculated for each menu item word, its weighted average ambiguity over the three dictionaries; that is, the number of senses a word has in each dictionary is weighted by the dictionary's dimension (Recall that the dimension of a dictionary is the total number of word senses it has.).

$$WA(w) = \frac{\sum_{k=1}^{N} n_k(w) \times d_k}{\sum_{k=1}^{N} d_k} \qquad (13)$$

where

$n_k(w)$ is the number of senses for w in dictionary k,
d_k is the dimension of dictionary k,
and
N is the number of dictionaries; in our case $N = 3$.

Because it is difficult to compare dictionaries, we attempt to homogenize the number of word senses over the dictionaries by weighting the number of senses for each word in each dictionary by the dimension of the dictionary. It is reasonable to assume that the more total word senses a dictionary has, the more senses it has per word. A graph showing the menu item words ranked by their weighted averages is shown in Figure 3. By weighted average ambiguity, the word cut is the most polysemous in each of the three dictionaries.

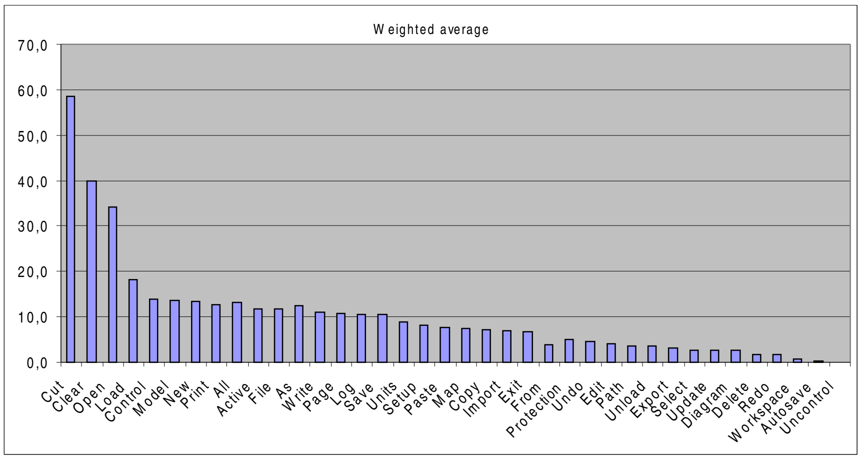


Fig. 3. Average Number of Word Senses per Menu Item in the Three Dictionaries Weighted by Dictionary Dimensions

A correlation coefficient provides a measure of linear association between variables. It is a value in the range $[-1, 1]$, where -1 means maximum negative linear correlation, 0 means no correlation, and 1 means maximum positive linear correlation. To determine the relationships between the dictionaries, we calculated the correlation coefficient for each pair of dictionaries. The correlation coefficient for a pair of dictionaries D_1 and D_2 is the correlation over all the menu-item words between the numbers of word senses in D_1 and D_2 for each menu-item word. Table 3 shows the three correlation coefficients between the three pairs of dictionaries. Each of the three values is quite high; thus,

WordNet – WordReference	WordNet – Babylon	WordReference – Babylon
0.85	0.63	0.83

Table 3. Correlations between the Numbers of Word Senses in the Dictionaries of each Menu Item Word

there is a high correlation between the dictionaries. The higher correlation coefficients between WordReference and the other two dictionaries can be explained by the WordReference dictionary's having a larger dimension than either of the other two. Recall the assumption that the more total word senses a dictionary has the more senses it has per word. These correlation coefficients between the dictionaries are only first approximations, because they are computed from the small set of command menu items. In order to have a more accurate estimate for correlation coefficients, it is necessary to use a larger corpus of words. The subsequent sections provide slightly larger corpora.

The graphs of Figures 2 and 3 show similar rankings of menu-item words. This similarity can be explained by the high correlation coefficients between the dictionaries. Indeed, the results of these experiments with $\mathrm{T1}_2$ say that the use of any of the three lexical resources yields pretty much the same output when the interest is to calculate lexical ambiguity. However, WordNet has an advantage that it is not just a dictionary. It is also a thesaurus, it provides synonyms and antonyms, and its synonym sets are interlinked by means of conceptual-semantic and lexical relations [37]. The other capabilities might prove to be useful in NLRSAI&M. Therefore, we decided to use WordNet as the lexical resource for the tool $\mathrm{T1}'$.

4.6 Prototyping of $\mathrm{T1}_3$

The purpose of constructing $\mathrm{T1}_3$ was to determine which of two possible auxiliary functions needed to calculate the lexical ambiguity of a sentence is the best for the purpose of NLRSAI&M. Lexical ambiguity at the sentence level, i.e., $\gamma(S)$, is used as a first approximation to a measure of the semantic ambiguity of a sentence. Ambiguities of words within a sentence combine in some way to give a total ambiguity of the sentence. The simplest method of combination is simply the sum of the lexical ambiguities of the sentence's words, as is the case for γ. However, other methods of combination are possible, including the product of the lexical ambiguities of the sentence's words or another more complex function of the lexical ambiguities of the sentence's words. The determination of which method of combination is best depends on the goal using lexical ambiguity at the sentence level as an approximation to the semantic ambiguity of a sentence.

The semantic ambiguity of S, $\mathit{SA}(S)$, can be expressed as a function F that depends on at least

- parameters of the dictionary D; these parameters include the total number of word senses in D, hereafter known as the *dimension of* D, and other characteristics, e.g., D's domain, and
- the lexical ambiguity $\alpha(S_i)$ of the word S_i of S, for each i in $[1..n]$.

$$\mathit{SA}(S) = F(P, \alpha(S_1), \ldots, \alpha(S_n), \ldots), \tag{14}$$

where

F is some function, and
P is the chosen parameter of D.

With this formula, $\gamma(S)$ can be obtained by letting F be $\sum$ and ignoring P.

In an effort to approximate semantic ambiguity, we experimented with F being "the sum of" and "the log base 2 of the product of". The idea is that "the sum of" F produces a lower bound, "the product of" F produces an upper bound, and "the log base 2 of the product of" F produces something in the middle. Moreover, the use of a logarithm in the definition of F is based on the definition of "the amount of information" in information theory [43]. It was necessary to conduct an experiment to determine which F is better. For now, let $\mathit{SA}_{\mathit{sum}}$ be the SA obtained when F is "the sum of", and let $\mathit{SA}_{\mathit{logProd}}$ be the SA obtained when F is "the log base 2 of the product of". Moreover, for any word w, let $\alpha(w)$ be calculated using the chosen D. We choose not to use any other parameters of D. Therefore,

$$\mathit{SA}_{\mathit{sum}}(S) = \sum_{j=1}^{n} \alpha(S_j) \tag{15}$$

$$\mathit{SA}_{\mathit{logProd}}(S) = \log_2 \prod_{j=1}^{n} \alpha(S_j) \tag{16}$$

We assumed that for any word, the probabilities of all its senses are the same. A more sophisticated measure of sentence ambiguity, such as α^* or α^S, could take into account that the frequencies of different senses of a word are different [4].

A problem with any measure that is a function of the number of senses of a word is that the number of senses of any word tends to be larger when determined using a larger dictionary. To normalize away the effect of any dictionary's size, each dictionary's contribution to the ambiguity measure is weighted by the dictionary's dimension, the total number of word senses the dictionary has. Therefore, the weighted sentence ambiguity of a sentence S, $\mathit{WSA}(S)$ is defined:

$$\mathit{WSA}(S) = \frac{\sum_{k=1}^{N} \mathit{SA}_{\mathit{meth},k}(S) \times d_k}{\sum_{k=1}^{N} d_k} \tag{17}$$

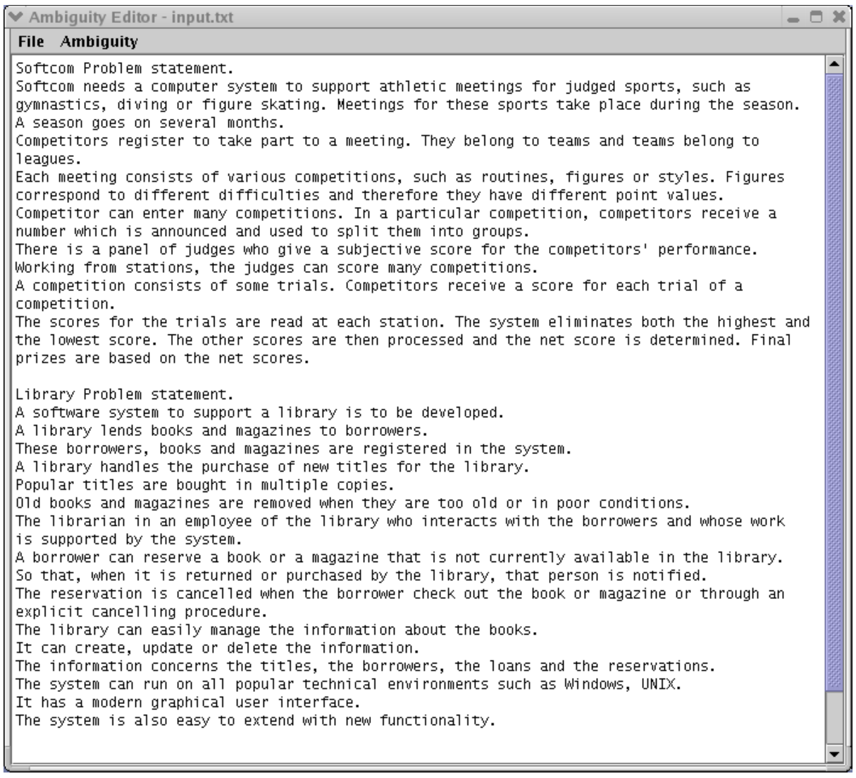
where

meth = *sum* or *logProd*,
$\mathit{SA}_{\mathit{meth},k}(S)$ is $\mathit{SA}_{\mathit{meth}}(S)$ (See Section 5.) calculated using dictionary k to count the number of senses of each of S's words,
d_k is the dimension of dictionary k, and
N is the number of dictionaries; in our case $N = 3$.

Therefore, T1$_3$ was built to compute both $\mathit{SA}_{\mathit{sum}}(S)$ and $\mathit{SA}_{\mathit{logProd}}(S)$ for each sentence S of its input, and the purpose of the experiment with T1$_3$ is to determine which of $\mathit{SA}_{\mathit{sum}}$ and $\mathit{SA}_{\mathit{logProd}}$ is most suitable for NLRSAI&M.

4.7 Experiment with T1_3

The artifacts processed in the experiment with T1_3 to determine which auxiliary function should be used to calculate γ are the two brief RSs, for the Softcom [44] and the Library [45] sytems, shown in Figure 4. Since the goal of these experiments was to learn



```
Softcom Problem statement.
Softcom needs a computer system to support athletic meetings for judged sports, such as
gymnastics, diving or figure skating. Meetings for these sports take place during the season.
A season goes on several months.
Competitors register to take part to a meeting. They belong to teams and teams belong to
leagues.
Each meeting consists of various competitions, such as routines, figures or styles. Figures
correspond to different difficulties and therefore they have different point values.
Competitor can enter many competitions. In a particular competition, competitors receive a
number which is announced and used to split them into groups.
There is a panel of judges who give a subjective score for the competitors' performance.
Working from stations, the judges can score many competitions.
A competition consists of some trials. Competitors receive a score for each trial of a
competition.
The scores for the trials are read at each station. The system eliminates both the highest and
the lowest score. The other scores are then processed and the net score is determined. Final
prizes are based on the net scores.

Library Problem statement.
A software system to support a library is to be developed.
A library lends books and magazines to borrowers.
These borrowers, books and magazines are registered in the system.
A library handles the purchase of new titles for the library.
Popular titles are bought in multiple copies.
Old books and magazines are removed when they are too old or in poor conditions.
The librarian in an employee of the library who interacts with the borrowers and whose work
is supported by the system.
A borrower can reserve a book or a magazine that is not currently available in the library.
So that, when it is returned or purchased by the library, that person is notified.
The reservation is cancelled when the borrower check out the book or magazine or through an
explicit cancelling procedure.
The library can easily manage the information about the books.
It can create, update or delete the information.
The information concerns the titles, the borrowers, the loans and the reservations.
The system can run on all popular technical environments such as Windows, UNIX.
It has a modern graphical user interface.
The system is also easy to extend with new functionality.
```

Fig. 4. RSs for SoftCom and Library Problems Loaded as Input

about the requirements for a tool to be built, the tool did not exist for the experiment. Therefore, we manually invoked existing commands to simulate what we thought the tool would do. We retrieved the number of senses for each word in the RSs in each of the three dictionaries and then calculated *WSA* for each sentence, including its stop words, in the RSs. We calculated also the average ambiguity of the sentences within each RS. Given that the WordReference dictionary has the highest dimension among the three dictionaries, it is no surprise that the sentence ambiguity measures calculated from the WordReference dictionary were the highest.

The calculations using $SA_{logProd}$ are summarized in Tables 4 and 5. In each of these tables, the data of each column is that described by the header of the column. In these headers, the dictionaries are identified by abbreviations to save horizontal space; "WR" means "WordReference", "WN" means "WordNet", and "B" means "Babylon". The rows are in the order of decreasing weighted average ambiguity, given in the last column. The last row, with "Avg." in the first column, gives for each column, the average value of the column's data for all of the rows above the last row. The results of the

Sentence Index in RS	No. Words in Sentence	No. Stop Words in Sentence	Lexical Ambiguity by WR	Lexical Ambiguity by WN	Lexical Ambiguity by B	Average Ambiguity	Weighted Average Ambiguity
10	19	6	63.8	33.0	28.0	41.6	43.5
8	17	6	62.0	33.2	24.2	39.8	41.7
6	15	4	50.7	35.6	32.3	39.5	40.5
7	21	9	58.9	23.7	28.2	36.9	38.7
9	15	4	51.6	24.3	28.1	34.7	36.1
14	13	2	45.6	26.1	27.6	33.1	34.1
1	11	4	39.8	27.6	17.9	28.4	29.4
4	11	5	41.6	19.8	18.5	26.6	27.8
3	10	3	34.5	20.5	16.3	23.8	24.7
16	10	3	31.2	18.8	18.7	22.9	23.6
11	10	3	34.6	18.6	14.5	22.5	23.6
13	12	6	40.6	13.1	12.1	21.9	23.5
5	7	1	23.5	16.7	16.0	18.7	19.1
2	8	3	28.6	13.1	11.8	17.8	18.7
15	7	1	22.4	14.5	12.0	16.3	16.8
12	8	1	16.3	12.5	9.2	12.7	13.0
Avg.	12.1	3.8	40.4	21.9	19.7	27.3	28.4

Table 4. $SA_{logProd}$ Calculation of Ambiguities of Sentences in Library RS

calculations using SA_{sum} are similar. The details of the calculations are given in Tables 10–17 and in the graphs of Figures 10–21 in Appendix 1.

Tables 4 and 5 show that the sentence ambiguities of the sentences in the two RSs are nearly the same. It is not possible to say at this time if these values are low or high. Because the two RSs have similar average sentence lengths, the similarity in their ambiguity values probably means only that the the two RSs have similar styles of writing, as any expert human reader could observe.

These sentence ambiguity functions provide only a high level, rough measure of sentential ambiguity. A more precise ambiguity measure would have to take into account the parts of speech. Taking the part of speech of a word into account reduces the apparent number of senses of word, as any sense consistent with a wrong part of speech need not be counted. We did, however, observe that the sentence ambiguity measures calculated by the two methods, by sum and by log base 2 of product, are quite similar to each other, in that the ratio of the values of different pairs of sentences in one RS are the

Sentence Index in RS	No. Words in Sentence	No. Stop Words in Sentence	Lexical Ambiguity by WR	Lexical Ambiguity by WN	Lexical Ambiguity by B	Average Ambiguity	Weighted Average Ambiguity
1	20	5	57.4	32.2	41.5	43.7	44.8
9	18	4	57.5	35.8	35.5	42.9	44.2
10	15	5	49.9	30.4	27.2	35.8	37.0
17	12	3	43.0	30.9	24.3	32.7	33.6
7	12	3	36.9	26.8	22.6	28.8	29.5
6	12	3	37.5	22.2	26.3	28.7	29.4
15	10	4	37.2	18.7	21.8	25.9	26.8
18	8	2	32.9	24.1	18.4	25.1	25.8
14	10	4	34.7	19.3	17.5	23.8	24.7
4	8	3	32.4	19.1	20.7	24.1	24.7
2	9	2	30.9	18.6	21.7	23.8	24.3
12	9	2	27.8	16.9	20.8	21.8	22.3
16	10	4	27.5	13.1	15.5	18.7	19.4
11	7	2	25.0	16.8	13.7	18.5	19.1
3	6	2	25.2	15.7	11.9	17.6	18.2
5	9	4	26.8	9.8	14.2	16.9	17.7
13	6	2	19.1	12.2	9.7	13.7	14.1
8	5	0	12.1	8.2	6.2	8.8	9.1
Avg.	10.3	3	34.1	20.6	20.5	25.1	25.8

Table 5. $SA_{logProd}$ Calculation of Ambiguities of Sentences in Softcom RS

same under the two methods of calculating the values. Thus, from the viewpoint of ambiguity values, the two methods are equivalent. Therefore, the more efficient method, summation, can be used whenever it is acceptable to approximate a sentence's semantic ambiguity by combining the lexical ambiguities of the words of the sentence.

Some additional lessons came from the experiments with $T1_3$. The calculated results confirm the existence of dependencies between: (1) the dimensions of dictionaries, (2) the number of words per sentence, (3) the lexical ambiguities of the words, and (4) the values of the ambiguity functions. For example, the data show that in each RS, there is a strong correlation between (1) the number of words in any sentence and (2) the sentence's weighted average ambiguity value, as well as between (1) the number of all words and the number of stop words in any sentence. Table 6 shows the two correlations in the Library RS, and Table 7 shows the two correlations in the Softcom RS. In fact,

Number of Words in a Sentence $\bowtie$ Weighted Average Ambiguity of the Sentence	Number of Words in a Sentence $\bowtie$ Number of Stop Words in the Sentence
0.92	0.85

Table 6. Correlations in Library RS

Number of Words in a Sentence $\bowtie$ Weighted Average Ambiguity of the Sentence	Number of Words in a Sentence $\bowtie$ Number of Stop Words in the Sentence
0.95	0.76

Table 7. Correlations in Softcom RS

any dependency between input parameters can be very complicated. These particular correlations seem to confirm the writing rule taught to students: "Shorter sentences are less ambiguous." Certainly a shorter sentence tends to be less lexically and syntactically ambiguous because it tends to have fewer total word senses and fewer parse trees than a longer sentence. However, pragmatically, a shorter sentence may say too little to allow its full meaning to be pinned down.

The high correlation between total words and stop words in a sentence says that stop words can be ignored in calculating sentence ambiguities.

4.8 Prototyping of $\mathrm{T1}_4$

The main goal of building $\mathrm{T1}_4$ was to test the effectiveness $\mathrm{T1}_4$'s way to compute the lexical ambiguity of sentences, which was determined by the experiments with $\mathrm{T1}_2$ and $\mathrm{T1}_3$. At the same time, $\mathrm{T1}_4$ was to be used to explore UI requirements for the basic $\mathrm{T1}'$.

4.9 $\mathrm{T1}_4$ Requirements

In the process of building the prototypes, $\mathrm{T1}_1$, $\mathrm{T1}_2$, and $\mathrm{T1}_3$, we were always thinking about UI requirements for the full tool, $\mathrm{T1}'$. In building $\mathrm{T1}_4$, we embodied the UI requirements as we understood them at that point so that the experiment with $\mathrm{T1}_4$ could be also a validation that the UI requirements were suitable.

The central part of the UI is the text area in which the user enters NL documents. This text area has to support all basic text processing functions which help in loading, editing, and saving a document in the file system. Some advanced text formatting features, such as changing type faces and sizes and giving the document structure, must be supported as well. This requirement can be achieved by letting the text area be a standard text-editor window.

The tool processes a RS in two phases. In the first phase, during the user's input of the RS, the tool detects and indicates what it believes are ambiguous words. In the second phase, which begins only after the user indicates that she is finished inputting the RS, the tool detects and indicates what it believes are ambiguous sentences. Note that what the tool believes is ambiguous may not indeed be ambiguous to any human reader, perhaps because he understands the disambiguating context. Therefore, whatever the tool believes is ambiguous is properly called only a "potential ambiguity".

During the first phase, which is during the user's input of the RS, the tool is parsing the input into words and sentences. As soon as the tool knows a word, it can look it up in its dictionary and calculate its lexical ambiguity. Thus, the user can be notified of a potentially ambiguous word as soon as she has entered the word.

Of course, the user cannot avoid using some common, ambiguous words. However, the meaning of each such word could and should be restricted to one domain-relevant sense appropriate for the CBS being specified. For example, each such word could be added to a local dictionary for the CBS that is accessed by the tool along with the global dictionary. Following the addition of a domain-relevant, otherwise ambiguous word to a local dictionary, the tool will no longer identify the word as ambiguous.

One possible feature of the tool is that the user is notified of a potential ambiguity in real time, just after the user has finished entering the potentially ambiguous text, while she is entering more text. This real-time notification has to be clear but not too heavy, because it should not interrupt the user during her writing. For example, a pop-up window or a sound are considered disturbing, because either can distract the user and can cause her to lose her train of thought. Changing the color of or underlining the potentially ambiguous word seems to be less distracting and more easily ignored if the user desires to continue thinking or entering text instead of looking immediately. On the other hand, if the user wants to deal with potential ambiguities immediately, the notification is visible peripherally, and she can turn her attention to it immediately.

During the second phase, the tool calculates the *SA* of each sentence of the RS. The tool notifies the user of a potentially ambiguous sentence S by changing the color of S's bounding box, i.e., S's immediate screen background. Green is reserved for low *SA* values; Red is reserved for high *SA* values; and Yellow is reserved for *SA* values that are neither low nor high. The meaning of the coloring is that the sentences with the red backgrounds need the greatest attention because they are potentially the most ambiguous sentences in the RS.

In $T1_4$, the user is asked to specify the highest *SA* value corresponding to green and lowest *SA* value corresponding to red. Since the coloring happens only after the tool has calculated and displayed the *SA* values for all sentences of a RS, the user has enough information to set these limit *SA* values for green and red according to her needs.

Perhaps, a future version of the tool can choose the coloring scale automatically: green would be reserved for the lowest *SA* value in the current RS; red would be reserved for the highest *SA* value in the RS; and a range of colors between green and red used for the increasing *SA* values in between the lowest and the highest. This automatically determined color scale may be useful when a user is approaching a RS for the first time and has no idea where the ambiguities may be. However, it has the drawback that the tool will always color some sentences in red in any RS no matter how carefully the user has rewritten the RS, unless the user manages to make every sentence exactly equally ambiguous, a highly unlikely event. Therefore, the tool should also allow the user to set the *SA*-to-color mapping to what she has determined is reasonable after a few rounds with a RS.

To reduce the ambiguity of a sentence S, the user must focus on the most ambiguous words in S, since the ambiguity of S is computed from the ambiguities of S's words. A good starting point for improving S is to change the potentially ambiguous words of S, i.e., the words that were colored or underlined during the first phase. After the user has dealt with the potentially ambiguous words in S, if $SA(S)$ remains high, the tool must help the user by suggesting other words to change.

As the user reduces the ambiguity of a sentence, in the worst case, she will need to change the whole sentence and write it in a different way. In other cases, the user simply changes individual words, replacing them by less ambiguous words. The tool should help the user by (1) providing her with a list of candidate less ambiguous substitute words or by (2) asking her if any words and their definitions should be added to the glossary being accumulated for the RS at hand.

The prototype tool is intended to facilitate writing of NL RSs. We expect that initially, a user will find herself being notified of many potential ambiguities. We expect also that as she learns the tool's behavior, she will begin to write RSs that are found to be less and less ambiguous by the tool.

4.10 Implementation and Behavior of $T1_4$

The prototype was implemented in the Java programming language. As determined by the experiment with $T1_2$, we used WordNet as the dictionary, and as determined by the experiment with $T1_3$, we used SA_{sum} to calculate γ.

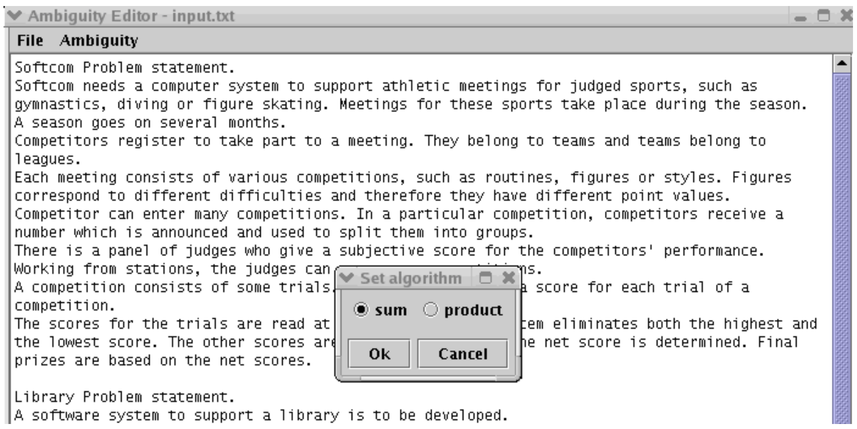
$T1_4$ uses a simple tokenizer to analyze the input. Since $T1_4$ calculates sentence ambiguity from only the lexical ambiguities of its words, $T1_4$ needs no syntactic parser.

If a parser will be needed for a future version of T1_4, the tokenizer will be reusable as providing the token sequence the parser needs.

4.11 Experiment with T1_4

This section demonstrates the T1_4's behavior on two RSs. First, the user has to open an input file containing at least one RS in plain text format. Figure 4 shows the result of loading a file containing two RSs, one titled "Softcom Problem Statement" and the other titled "Library Problem Statement".

Next, the user must choose the function for computing the ambiguity of sentences. So far the choice is between the sum and the logarithm-base-two-of-the-product functions described in Section 5. Figure 5 shows the actual choice window in which the sum function has been selected.



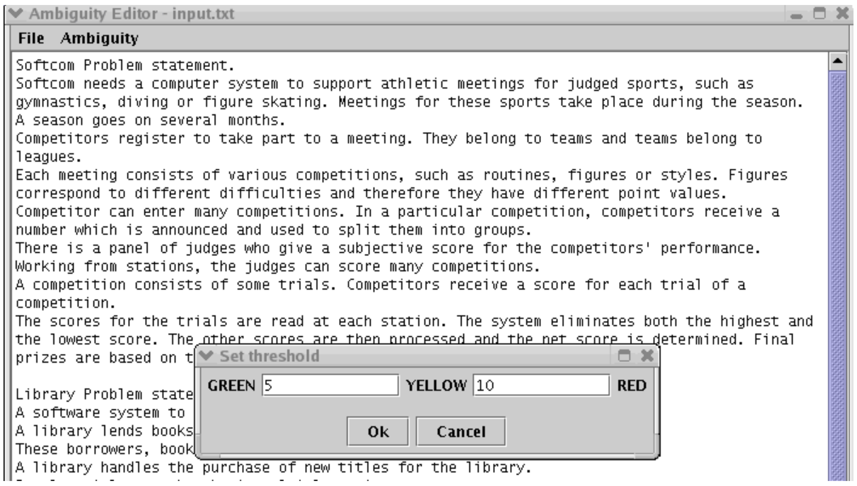
```
Softcom Problem statement.
Softcom needs a computer system to support athletic meetings for judged sports, such as
gymnastics, diving or figure skating. Meetings for these sports take place during the season.
A season goes on several months.
Competitors register to take part to a meeting. They belong to teams and teams belong to
leagues.
Each meeting consists of various competitions, such as routines, figures or styles. Figures
correspond to different difficulties and therefore they have different point values.
Competitor can enter many competitions. In a particular competition, competitors receive a
number which is announced and used to split them into groups.
There is a panel of judges who give a subjective score for the competitors' performance.
Working from stations, the judges can                    ns.
A competition consists of some trials.                   a score for each trial of a
competition.
The scores for the trials are read at                    tem eliminates both the highest and
the lowest score. The other scores are                   e net score is determined. Final
prizes are based on the net scores.

Library Problem statement.
A software system to support a library is to be developed.
```

[● sum] ○ product

Fig. 5. Choosing the Function for Calculating Sentence Ambiguity

As mentioned, the number of the senses for each word is determined from the WordNet database index. The tool accesses the database at run time as it needs to. The user can request the tool to recalculate measures at any time, e.g., after changing the input text or after adding another word to the local dictionary.

Then, the user is given the opportunity to decide on the mapping of sentence ambiguity values to the colors that the tool uses to notify the user of potentially ambiguous sentences. Figure 6 shows the window in which three ambiguity value ranges are associated with three colors. In this window, the ambiguity values less than or equal to 5 are assigned the green color, and the ambiguity values greater than or equal to 10 are assigned the red color; implicitly, ambiguity values greater than 5 but less than 10 are assigned the yellow color.[4]



```
Softcom Problem statement.
Softcom needs a computer system to support athletic meetings for judged sports, such as
gymnastics, diving or figure skating. Meetings for these sports take place during the season.
A season goes on several months.
Competitors register to take part to a meeting. They belong to teams and teams belong to
leagues.
Each meeting consists of various competitions, such as routines, figures or styles. Figures
correspond to different difficulties and therefore they have different point values.
Competitor can enter many competitions. In a particular competition, competitors receive a
number which is announced and used to split them into groups.
There is a panel of judges who give a subjective score for the competitors' performance.
Working from stations, the judges can score many competitions.
A competition consists of some trials. Competitors receive a score for each trial of a
competition.
The scores for the trials are read at each station. The system eliminates both the highest and
the lowest score. The other scores are then processed and the net score is determined. Final
prizes are based on t

Library Problem state
A software system to
A library lends books
These borrowers, book
A library handles the purchase of new titles for the library.
```

GREEN 5 YELLOW 10 RED

Fig. 6. Setting Ambiguity-Value–Color Mapping

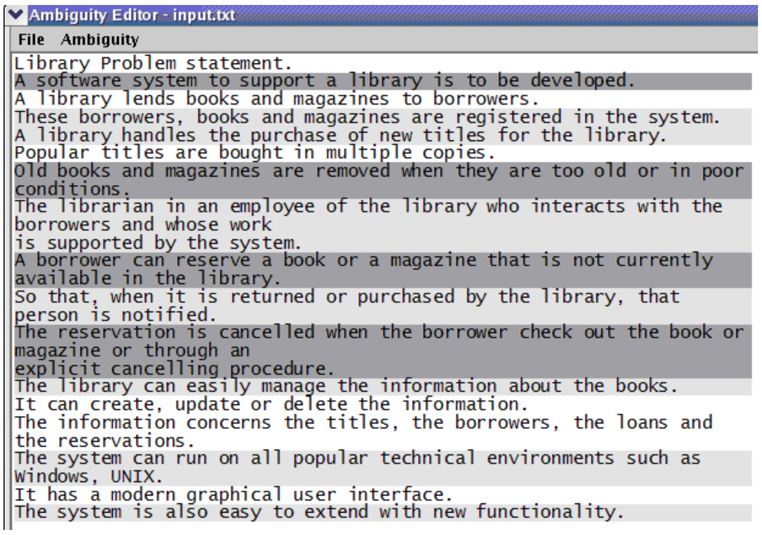
The tool's final output is the input RSs in which the bounding box of each sentence is colored as described above, according to the sentence's *SA* calculated with the selected function. Figure 7 shows the output of the Library Problem Statement, and Figure 8 shows the output of the Softcom Problem Statement.

4.12 Additional Experiment with $\mathrm{T1}_4$

We applied the prototype tool $\mathrm{T1}_4$, with the same function choice and the same ambiguity-value–color mapping, to the ABCVPS, described in Section 4, which is about T1. Figure 9 shows the resulting output.

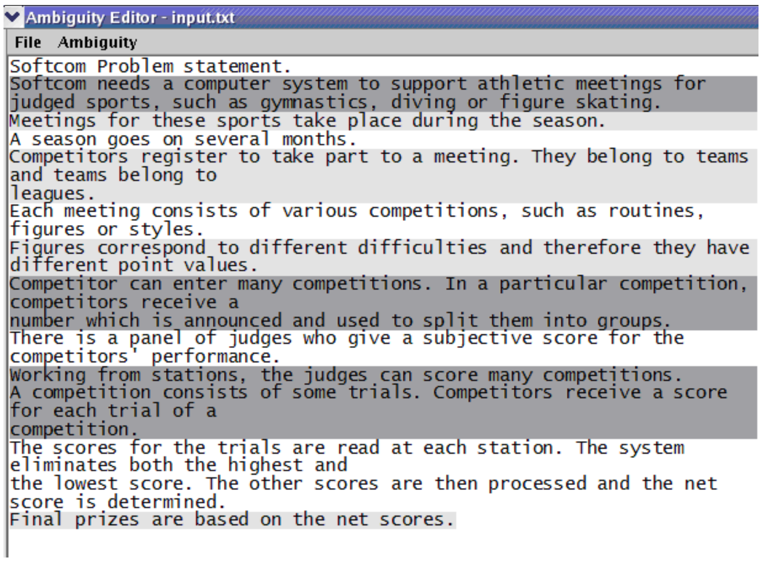
It is interesting to compare the sentences marked as highly ambiguous in Figure 9 with the sentences of the same input marked as highly ambiguous, in Table 1, by T1. Recall that in Table 1, first column gives the penalty-weighted ambiguity, the δ^*, of the sentence in the row and that in Figure 9, the color red means "highly ambiguous", the color yellow means "somewhat ambiguous", and green means "little or no ambiguity". While every sentence that is given a δ^* greater than 30 by T1 is marked red by $\mathrm{T2}_3$, the opposite is decidedly not true. A sentence marked red by $\mathrm{T2}_3$ is given almost any δ^* by T1. Each sentence given a δ^* of only 1 or 2 by T1 is marked either yellow or red by $\mathrm{T2}_3$.

[4] Figures 7, 8, and 9 use this color scheme. If you are reading a black-and-white printing of the paper, the printer substitutes white for green, light gray for yellow, and dark gray for red. Thus, the potentially most ambiguous sentences are those with a dark gray background.



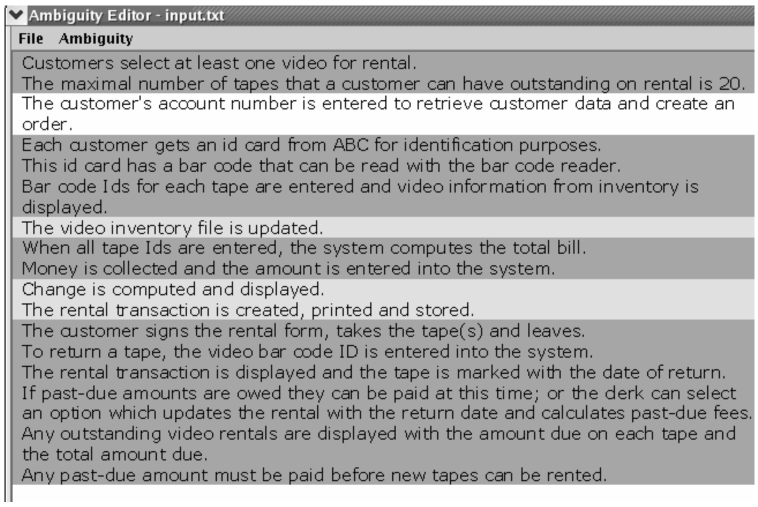
```
Library Problem statement.
A software system to support a library is to be developed.
A library lends books and magazines to borrowers.
These borrowers, books and magazines are registered in the system.
A library handles the purchase of new titles for the library.
Popular titles are bought in multiple copies.
Old books and magazines are removed when they are too old or in poor
conditions.
The librarian in an employee of the library who interacts with the
borrowers and whose work
is supported by the system.
A borrower can reserve a book or a magazine that is not currently
available in the library.
So that, when it is returned or purchased by the library, that
person is notified.
The reservation is cancelled when the borrower check out the book or
magazine or through an
explicit cancelling procedure.
The library can easily manage the information about the books.
It can create, update or delete the information.
The information concerns the titles, the borrowers, the loans and
the reservations.
The system can run on all popular technical environments such as
Windows, UNIX.
It has a modern graphical user interface.
The system is also easy to extend with new functionality.
```

Fig. 7. Output of Library Problem Statement with Sentences Colored According to their Sentence Lexical Ambiguity (*SA*) Values



```
Softcom Problem statement.
Softcom needs a computer system to support athletic meetings for
judged sports, such as gymnastics, diving or figure skating.
Meetings for these sports take place during the season.
A season goes on several months.
Competitors register to take part to a meeting. They belong to teams
and teams belong to
leagues.
Each meeting consists of various competitions, such as routines,
figures or styles.
Figures correspond to different difficulties and therefore they have
different point values.
Competitor can enter many competitions. In a particular competition,
competitors receive a
number which is announced and used to split them into groups.
There is a panel of judges who give a subjective score for the
competitors' performance.
Working from stations, the judges can score many competitions.
A competition consists of some trials. Competitors receive a score
for each trial of a
competition.
The scores for the trials are read at each station. The system
eliminates both the highest and
the lowest score. The other scores are then processed and the net
score is determined.
Final prizes are based on the net scores.
```

Fig. 8. Output of Softcom Problem Statement with Sentences Colored According to their Sentence Lexical Ambiguity (*SA*) Values



Customers select at least one video for rental.
The maximal number of tapes that a customer can have outstanding on rental is 20.
The customer's account number is entered to retrieve customer data and create an order.
Each customer gets an id card from ABC for identification purposes.
This id card has a bar code that can be read with the bar code reader.
Bar code Ids for each tape are entered and video information from inventory is displayed.
The video inventory file is updated.
When all tape Ids are entered, the system computes the total bill.
Money is collected and the amount is entered into the system.
Change is computed and displayed.
The rental transaction is created, printed and stored.
The customer signs the rental form, takes the tape(s) and leaves.
To return a tape, the video bar code ID is entered into the system.
The rental transaction is displayed and the tape is marked with the date of return.
If past-due amounts are owed they can be paid at this time; or the clerk can select an option which updates the rental with the return date and calculates past-due fees.
Any outstanding video rentals are displayed with the amount due on each tape and the total amount due.
Any past-due amount must be paid before new tapes can be rented.

Fig. 9. Output of ABC Video Problem Statement with Sentences Colored According to their Sentence Lexical Ambiguity (*SA*) Values

The only sentence marked green by $\mathrm{T2}_3$ is given a δ^* of 14 by T1. Thus, there is very little agreement between the tools' conceptions of which sentences are at any level of ambiguity, even though they are computing the same function $\delta = SA_{sum}$, albeit from different dictionaries. Moreover,

1. there is no sentence of the ABCVPS that is not determined to be at least some what ambiguous by at least one of the tools and
2. every sentence but three is marked at least highly ambiguous by at least one of the tools.

None of the sentences is considered uniguous by both tools.

4.13 Conclusions of Experiments with $\mathrm{T1}_4$

$\mathrm{T1}_4$ proved to be effective in computing the lexical-ambiguity functions of $\mathrm{T1}'$. $\mathrm{T1}_4$ is built using only publically accessible resources. In addition, the UI that we put into $\mathrm{T1}_4$ seems to be helpful.

5 Requirements for T2

At the very least, T2 could exhibit for any sentence, all of its parse trees and all the word senses for each of its words, and it could get this information from T1. However, this output is not enough. There are serious problems with the ABCVPS that are not detected by the current T1. The purpose of this section is to identify other ambiguity problems that should be identified and exhibited by T2 when presented with a RS.

5.1 Experiment with T2

To learn what these other ambiguities are, we manually examined the ABCVPS to search for instances of a variety of problems mentioned in a variety of sources, including work by:

- Berry, Kamsties, and Krieger on ambiguities in NL RSs and legal contracts [12],
- Berry and Kamsties on the syntactically and semantically dangerous "all" and plural [12, 46],
- Bucchiarone, Fabbrini, Fusani, Gnesi, Lami, Pierini, and Trentanni on a model of the quality of RSs [3, 47],
- Denger on rules and patterns for high-quality RSs [30],
- Dupré on technical writing [48],
- Fuchs, Schwitter, and Schwertel on controlled English [32, 33],
- Kovitz on the style of RSs [49], and
- Rupp and Goetz on Neurolinguistic Processing [50].

Not all problems mentioned in these sources appear in the ABCVPS.

The list below gives the sentences of the ABCVPS. Each list item gives the sentence followed by an enumeration of the problems found in the sentence. Each problematic phrase[5] is bracketed and each pair of bracket has an index referring to an item in the enumeration of the problems in the sentence. Not all items in the enumeration of problems for a sentence are referred to by a bracket pair's index. Each nonreferenced item is a question involving more than one phrase or sentence. Only the first time a particular problem occurs, a detailed explanation of the problem is given, surrounded by "≪" and "≫". In such an explanation, example text from the sentence whose problem is being explained is said to be from "the sentence at hand" so that examples from elsewhere can be addressed as "the example".

1. [Customers]$^{(a)}$ select at least one video for rental.
   (a) Plural subject: ≪The problem with a plural subject is that in the absence of domain knowledge, it is not clear whether the complement of the verb applies to each instance of the subject or to plural subject as a whole [51, 46, 33]. That is, in the sentence at hand, does each customer select at least one video for rental or do customers together select at least one video for rental. Perhaps, a clearer example is the sentence Three girls lift one table. [33] Does each of the three girls lift one table on her own or do all three girls lift one table together? The solution to the problem is to use only singular subjects; if the intent *is* to say that the plural subject does the action of the verb, then a singular collective noun should be used as the subject. Thus the example would be written as either Each of three girls lifts on table. or A group of three girls lifts one table.≫ For the sentence at hand, domain knowledge tells us that the intent is probably to say Each customer selects at least one video for rental.
2. The maximal number of tapes that a customer can have outstanding on rental is 20.

[5] "Phrase" is used in this section for "word or phrase" since a word is a degenerate phrase.

(a) There is nothing in the ABCVPS that says that video and tape are synonyms. Domain knowledge tells us that they probably are synonyms. Moreover, the sentence at hand says that tapes can be outstanding on rental, while Sentence 16 of the ABCVPS talks about outstanding video rentals. ≪The problem with the presence of synonyms in a RS is that without domain knowledge, the reader cannot know for certain that the synonyms mean the same. The reader is left believing that or wondering if the denotations of the synonyms are different. The problem is far more severe in an industrial strength RS written by several different people, each with his own set of synonyms for a concept. The solution is to decide on one term for each concept, that is, one representative from among each set of synonyms, and to use only that term or representative. Why do people use synonyms? Perhaps, they use synonyms from a misplaced goal of keeping the writing interesting. Perhaps, a RS with lots of synonyms is more interesting, but at the cost of being more ambiguous and confusing. The excitement of resolving the ambiguities and confusion, of tracking down all the synonyms might add to the interest, but at what cost?≫

3. The customer's account number [is](a) entered to retrieve customer data and create an order.
   (a) Passive voice: ≪The problem with passive voice is that in the absence of domain knowledge, it is not clear who or what is doing the action [52, 50]. The most problematic implication of this lack of clarity is that it is not even clear whether (1) the environment does the action to the CBS or (2) the CBS does the action. In the former case, the requirement is for the CBS to *react to* the action. In the latter case, the requirement is for the CBS to *do* the action. This distinction is critical for writing the CBS's requirements correctly. The sentence should be rewritten in active voice with an explicit subject doing the action.≫ The sentence at hand is truly ambiguous, because domain knowledge suggests that either the customer, an employee of ABC Video, or both could enter the customer's account number, e.g., by swiping the customer's id card in a bar code reader. Therefore, the requirements engineer would have to consult the customer about his or her desires in order to disambiguate the sentence at hand in order to rewrite it in active voice! An arbitrary choice has to be made for the rewritten ABCVPS presented in Section 8.2.
4. Each customer gets an id card from ABC for [identification purposes](a).
   (a) Weak phrase: ≪The problem with a weak phrase is that in the absence of domain knowledge, it is not clear what the phrase implies for the requirements of the CBS at hand [49].≫ In this case, what are the identification purposes? The solution is to replace the weak phrase with a more detailed phrase. The most likely meaning of for identification purposes in the sentence at hand is to identify the customer that is the subject of the sentence. Sometimes the replaced phrase has more text than just the weak phrase itself.
   (b) There is nothing that says that a customer gets only one id card from ABC. The sentence at hand says only that each customer gets an id card from ABC, and says nothing about making sure that a customer does not get more than one id card from ABC. Therefore, the ABC System really needs to allow for a customer to have more than one id card.

5. This id card has a bar code that can be read with [the][(a)] bar code reader.
   (a) Noun with definite article not introduced before: ≪The meaning of a noun preceded by a definite article, i.e., the, is that there is an instance of the denotation of the noun introduced in a previous sentence, by name or by use of an indefinite article, i.e., a, and that the instance with the definite article refers to that previously introduced instance [48].≫ The sentence at hand has the phrase the bar code reader. The question is “To what bar code reader is the phrase referring?” None has been introduced in any previous sentence within the ABCVPS. Probably, the intent of the author of the sentence was to simultaneously introduce a bar code reader and to say that there is only one. The most direct way to achieve this intent is to say The ABC system has one bar code reader. From that sentence on, it is legitimate to talk about the bar code reader. If the uniqueness of the bar code reader is not required, then the author should say only The ABC system has a bar code reader. From that sentence on, it is legitimate to talk about the bar code reader, but meaning only the one mentioned before.
   (b) There is nothing in the sentence at hand or even in the entire ABCVPS that relates a customer’s account number to the bar code of an id card that the customer has. Domain knowledge suggests that the bar code of an id card that a customer probably contains an account number for the customer, and thus, the bar code of an id card for a customer and an account number for a customer are probably synonyms.
6. Bar code Ids for each tape [are][(a)] entered and video information from inventory [is][(b)] displayed.
   (a) Passive voice: Who or what enters bar code ids for each tape? A reasonable answer is an employee of ABC Video. However, with an automated system, the customer could very well enter bar code ids for each tape himself, by waving each tape in front of a bar code reader. Only the future owner of the ABC System can answer the question. For this example, there is no real future owner that we can ask, so we make an arbitrary choice that the answer is an employee of ABC Video. Indeed, for any such question that can be answered by only the future owner, we provide an arbitrary, reasonable answer.
   (b) Passive voice: Who or what displays video information from inventory? The most likely answer is the ABC System, which is the subject of the ABCVPS RS.
7. [The][(a)] video inventory file [is][(b)] updated.
   (a) Noun with definite article not introduced before: What video inventory file? If, as suggested in Item (c) below, video information from inventory and video inventory file are synonyms, then The video inventory file is the previously introduced video information from inventory
   (b) Passive voice: Who or what updates the video inventory file? The most likely answer is the ABC System, which is the subject of the ABCVPS RS.
   (c) Apparently, video information from inventory and video inventory file are synonyms.
8. When all tape Ids [are][(a)] entered, [the][(b)] system computes the total bill.

   (a) Passive voice: Who or what enters all tape Ids? A reasonable answer is an employee of ABC Video.
   (b) Noun with definite article not introduced before: What system? The most likely answer is the ABC system that is the subject of the ABCVPS RS.

9. Money [is]$^{(a)}$ collected and the amount [is]$^{(b)}$ entered into the$^{(c)}$ system.
   (a) Passive voice: Who or what collects money? A reasonable answer is an employee of ABC Video.
   (b) Passive voice: Who or what enters the amount into the system? A reasonable answer is an employee of ABC Video.
   (c) This instance of a definite article is *not* bracketed because the system was introduced in the previous sentence.
   (d) What is the relationship between money and amount? Domain knowledge suggests that amount is a property of money.
10. Change [is]$^{(a)}$ computed and displayed.
   (a) Passive voice: Who or what computes and displays change? The most likely answer is the ABC System, which is the subject of the ABCVPS RS.
   (b) What is the relationship between change and what has appeared before? Domain knowledge suggests that change is the arithmetic difference between the amount of money collected and the total bill.
11. [The]$^{(a)}$ rental transaction [is]$^{(b)}$ created, printed and stored.
   (a) Noun with definite article not introduced before: What rental transaction? The most likely answer is that the rental transaction is the unique rental transaction being created, printed, and stored in the sentence at hand. In this case, *a* rental transaction is being created, printed, and stored.
   (b) Passive voice: Who or what creates, prints, and stores the rental transaction? The most likely answer is the ABC System, which is the subject of the ABCVPS RS.
12. The customer signs [the]$^{(a)}$ rental form, takes the tape(s) and leaves.
   (a) Noun with definite article not introduced before: What rental form? The most likely answer is that the rental form is the rental transaction that is printed in the previous sentence and that rental form is a synonym for printed rental transaction.
   (b) Apparently, rental form and printed rental transaction are synonyms.
13. To return a tape, the video bar code ID [is]$^{(a)}$ entered into the system.
   (a) Passive voice: Who or what enters the video bar code ID into the system? Reasonable answers are the customer returning the tape and an employee of ABC Video. An arbitrary choice has to be made for the rewritten ABCVPS.
   (b) Apparently, video and tape *are* synonyms because both words are used in the same sentence in a way that indicates that they are synonyms.
14. The rental transaction [is]$^{(a)}$ displayed and the tape [is]$^{(b)}$ marked with the date of return.
   (a) Passive voice: Who or what displays the rental transaction? The most likely answer is the ABC System, which is the subject of the ABCVPS RS.
   (b) Passive voice: Who or what marks the tape with the date of return? The most likely answer is the ABC System, which is the subject of the ABCVPS RS.

   (c) The physical tape is marked with the date of return? Domain knowledge suggests that the physical tape is not marked; rather the video information from inventory for the tape is changed to show the date of return.

15. If past-due amounts [are][(a)] owed they[(d)] can [be][(b)] paid at this time; or [the][(c)] clerk can select an option which updates the rental with the return date and calculates past-due fees.
   (a) Passive voice: Who or what owes past-due amounts? Domain knowledge suggests that the most likely answer is the customer.
   (b) Passive voice: Who or what can pay them, i.e., the past-due amounts, at this time? Domain knowledge suggests that the most likely answer is the customer who likely owes past-due amounts.
   (c) Noun with definite article not introduced before: What clerk? There is no clerk mentioned before. Domain knowledge suggests that the clerk that was suddenly introduced in the sentence at hand is the mysterious employee of ABC Video that we had to invent to actively do the clerical functions of ABC Video that are expressed in passive voice.
   (d) The they is *not* bracketed because it clearly refers to the immediately preceding plural noun phrase past-due amounts
   (e) Should not the or following the semicolon be and?
   (f) Are amounts and fees synonyms? After all, each can be past due. Domain knowledge suggests that indeed amounts and fees *are* synonyms.
   (g) Both amount and amounts appear, the second being the plural of the first.
   (h) Apparently clerk and employee of ABC Video are synonyms.
16. Any outstanding video rentals [are][(a)] displayed with the amount due on each tape and the total amount due.
   (a) Passive voice: Who or what displays any outstanding video rentals with the amount due and the total amount due? The most likely answer is the ABC System, which is the subject of the ABCVPS RS.
17. Any past-due amount must [be][(a)] paid before new tapes can [be][(b)] rented.
   (a) Passive voice: Who or what must pay any past-due amounts before new tapes can be rented? The most likely answer is the customer.
   (b) Passive voice: Who or what can rent new tapes? The most likely answer is the customer.
   (c) What is the relationship between past-due amount and amount due on a tape and total amount due? Domain knowledge suggests that the sum of first two equals the latter.

The most common problems were (1) the presence of passive voice, (2) the presence of definite articles with no referents, and (3) the use of synonyms. In this very small problem statement in a very familiar domain, the discovery of synonyms is manageable. However, in a large problem statement or in an esoteric domain, the discovery of synonyms is highly error prone.

Instances of Problems 1 and 2 and problems similar to them require T2 to have access to parse trees, parts of speech information, and other structural information about the sentences of T2's input RS. If also T2 were built based on LOLITA, this information would already be available from having run T1 on the same RS.

Handling problem 3, requires discovery of synonyms. If T2 were based on LOLITA, then T2 would have access to a semantic net. The semantic net combined with the use of a thesaurus, such as the Web-accessible WordNet [37], offers a hope of automating the discovery of synonyms. Of course, the human user would have to be asked to confirm that any pair of automatically discovered synonym is indeed a pair of real synonyms.

Finally, the functionality of T1 has to be changed so that it finds these new kinds of ambiguity and it uses some measure of the severity of each instance of these new kinds of ambiguity in computing the level of ambiguity of each sentence in a RS presented to T1.

5.2 Rewritten ABCVPS

This subsection shows the ABCVPS rewritten to remove all the problems mentioned in the previous subsection. The the ABCVPS is completely rewritten into three scenarios preceded by three indicative statements and one invariant statement. Each sentence derived from the original ABCVPS has been rewritten to avoid all the problems identified in this subsection. In particular each sentence is rewritten into active voice with a singular subject. Throughout all sentences, any word which has synonyms is replaced by a single chosen representative of each set of synonyms.

Therefore the first step was to identify the sets of synonyms and to choose the representative from among the elements of the set that is to be used in any rewrite of the ABCVPS. Synonym identification is combined with identification of all terms in the ABCVPS and the fusion of any multiword term into a single token with "_"s replacing the spaces between the words. Table 8 shows each term occurring in the original ABCVPS and the term designated to replace it. Any table row that has more than one original term is for a set of synonym terms; that row has only one replacement term. The rows are alphabetized by the original terms. A row with a synonym set appears once for each member of the set in the proper positions in the alphabetical ordering of the original terms, *unless the repeated rows are adjacent to each other in the ordering*. Likewise, a row whose term begins with a stop word, i.e., a, an, for, or the, appears twice, once in the ordering according to the stop word and once in the ordering according to the word following the stop word *unless the repeated rows are adjacent to each other in the ordering*.

Converting to active voice led to the discovery of two terms not in the original ABCVPS, for hidden actors that do the actions that were expressed passively.

– ABC_system
– clerk

In retrospect, it is clear that the ABCVPS is not a RS but is three scenarios of optative [53] sentences with some global indicative and invariant sentences about a CBS called the ABC System. In particular,

– Each of Sentences 4 and 5 is an indicative statement about the ABC Video world independent of the ABC System, something that is true even in a completely manual version of ABC Video's business.

Original Term	**Replacement Term**
ABC	ABC
an account number [for a customer] the bar code of an id card [for a customer]	id_card's bar_code
amount [of money]	sum [of money]
amounts fees past-due amounts	amount_due
an account number [for a customer] the bar code of an id card [for a customer]	id_card's bar_code
bar code	bar_code
the bar code of an id card [for a customer] an account number [for a customer]	id_card's bar_code
bar code reader	bar_code_reader
clerk employee of ABC Video	clerk
customer	customer
customer data	customer_data
employee of ABC Video clerk	clerk
for identification purposes	that identifies the customer
id card	id_card
maximal	maximum
money	money
order	order
outstanding on rental outstanding video rentals	on_rental
past-due amount	amount_due
past-due amounts amounts fees	amounts_due
printed rental transaction rental form	rental_transaction
rental transaction	order
tape video	video_tape
the bar code of an id card [for a customer] an account number [for a customer]	id_card's bar_code
total bill	total_bill
video information from inventory video inventory file	video_information from the video_inventory
video tape	video_tape

Table 8. Replacement Terms Including for Synonyms

- Sentence 2 is a global invariant sentence about the ABC System, a property that must be maintained as true by any transaction of the ABS System.
- Sentences 1, 3, 17, 6, 7, 8, 9, 10, 11, and 12, in that order, describe the steps of Scenario 1, **Customer rents at least one video tape**.
- Sentences 13, 14, and 16 in that order, describe the steps of Scenario 2, **Customer returns at least one video tape**.
- Sentence 16 describes the steps of Scenario 3, **Customer pays his amount due** that can be used by other scenarios.

For each rewritten sentence, the bracketed number typeset in a roman serifed typeface at the end of the rewritten sentence is the index in the original ABCVPS of the sentence from which the rewritten sentence is derived. Each noun representing a datum, whether occurring directly in the original ABCVPS or being the representative of synonyms occurring in the original ABCVPS is typeset in a slanted typeface. Each comment about the production of the rewritten ABCVPS is typeset in an italic serifed typeface, to distinguish it from both sentences that are part of the rewritten ABCVPS and ordinary text.

In creating this rewritten ABCVPS, we are disambiguating many an ambiguity by making assumptions about the domain explicit. Many of these assumptions should be made by only the future owner of the ABC System. For this example, there is no real future owner; therefore, we make arbitrary, but reasonable assumptions. However, for any real project, identification of any such ambiguity should be taken as a prompt for the analysts to gather additional information for disambiguation from stakeholders rather than for the analysts to disambiguate themselves on the basis of possibly incorrect assumptions.

Indicative Statements about ABC Video Independent of ABC System

Each *customer* gets from *ABC* an *id_card* that identifies the *customer*. [4]

Each *id_card* has a unique *bar_code* that any *bar_code_reader*
can read. [5]

Each *customer* *bar_code* indexes at most one datum in the *ABC_system*'s *customer_database*. [new]

The bar_code of an id_card of a customer is an account_number for the customer. NOTE: We will not use account_number at all, sticking with bar_code.

We cannot guarantee a unique account_number for any customer, because while bar_codes, and therefore account_numbers are unique among id_cards, no customer is excluded from getting more than one id_card.

Each *video_tape bar_code* indexes at most one datum in the *ABC_system*'s *video_inventory*. [new]

Global Invariant about ABC System

The maximum number of *video_tape*s that any *customer* can have *on_rental* at any time is 20. [2]

Scenario 1: Customer rents at least one *video_tape*.

A *customer* selects at least one *video_tape* to rent from the *ABC_system*. [1]

The *customer* shows one of his *id_card*s to one of the *ABC_system*'s *bar_code_reader*s. [3]

The *ABC_system* reads the *id_card*'s *bar_code* through the *bar_code_reader*. [3]

The *ABC_system* retrieves *customer_data* as the datum that the *id_card*'s *bar_code* indexes in the *ABC_system*'s *customer_database*. [3]

The *ABC_system* displays the *customer_data*. [3]

The *ABC_system* creates an *order* that is for the *id_card*'s *bar_code*. [3]

If the *customer_data* shows any *amount_due*, then [17]

the *ABC_system* informs the *customer* that he must pay the *amount_due* before he can rent any more video tapes; and stop. [17]

The *ABC_system* sets to 0 the *total_bill* of the *order*. [6]

For each *video_tape* selected by the *customer*: [6]

The *customer* shows the *video_tape* to the *bar_code_reader*. [6]

The *ABC_system* reads the *video_tape*'s *bar_code* through the *bar_code_reader*. [6]

The *ABC_system* retrieves *video_information* as the datum that the *video_tape*'s *bar_code* indexes in the *ABC_system*'s *video_inventory*. [6]

The *ABC_system* displays the *video_information*. [6]

The *ABC_system* sets to *on_rental* the *video_information*. [7]

The *ABC_system* copies the *video_information* to the *order* [7]

The *ABC_system* sets to *today's_date* plus 7 days the *due_date* for the *video_tape*'s *bar_code* in the *order*. [7]

The *ABC_system* adds the *rental_fee* of the *video_information* to the *total_bill* of the *order*. [8]

The *customer* gives *money* to a *clerk*. [9]

The *clerk* computes the *sum* of the *money*. [9]

If the *sum* is greater than the *total_bill*, then [10]

the *clerk* gives the *sum* - *total_bill* as change to the *customer*. [10]

The *clerk* instructs the *ABC_system* to print the *order* twice. [11]

The *ABC_system* prints the *order* twice as *rental_transaction*s. [11]

The *ABC_system* stores the *order* that the *id_card*'s *bar_code* indexes in the *ABC_system*'s *rental_database*. [11]

```
The customer signs one rental_transaction. [12]
The customer gives the signed rental_transaction to the clerk. [12]
The customer leaves, taking the other rental_transaction and the
    video_tapes. [12]
```

Scenario 2: Customer returns at least one *video_tape*.

```
The customer shows one of his id_cards to one of the ABC_system's
    bar_code_readers. [13]
The ABC_system reads the id_card's bar_code through the
    bar_code_reader. [13]
The ABC_system retrieves customer_data as the datum that the id_card's
    bar_code indexes in the ABC_system's customer_database. [13]
The ABC_system displays the customer_data. [13]
For each video_tape that is being returned by the customer: [13]
    The customer shows the video_tape to the bar_code_reader. [13]
    The ABC_system reads the video_tape's bar_code through the
        bar_code_reader. [13]
    The ABC_system retrieves video_information as the datum that the video_tape's
        bar_code indexes in the ABC_system's video_inventory. [13]
    The ABC_system displays the video_information. [13]
    The ABC_system sets to in_store the video_information as of today's_date.
        [14]
    The ABC_system retrieves order as the datum that the video_tape's
        bar_code indexes in the ABC_system's rental_database. [14]
    The ABC_system displays the order. [14]
    If the due_date of the video_tape's bar_code is before today's_date,
        then [14]
        the ABC_system calculates a fine as (today's_date - due_date) *
            $1.00; [14]
        the ABC_system adds the fine to the amount_due of the customer_data.
            [14]
    The ABC_system removes the order from the ABC_system's rental_database.
        [14]
    The ABC_system removes the video_tape's bar_code from the
        customer_data. [14]
For each video_tape's bar_code in customer_data, that of a non-returned
video_tape: [16]
    The ABC_system displays the video_tape's bar_code. [16]
    The ABC_system computes the fine for the video_tape's bar_code as
        (today's_date - due_date of the video_tape's bar_code) *
        $1.00. [16]
    The ABC_system displays as a warning the fine for the video_tape's
        bar_code. [16]
The ABC_system displays the amount_due of the customer_data. [16]
```

Scenario 3: Customer pays his *amount_due.*

S3 is a subscenario of S1 and S2.

Since Scenario 3 is a subscenario of S1 and S2, it is assumed that a customer_data is available identifying the customer who must pay his amount_due.

```
The ABC_system displays the amount_due of the customer_data. [15]
If the amount_due is greater than $0.00, then [15]
    the customer gives money to a clerk. [15]
    The clerk computes the sum of the money. [15]
    If the sum is greater than the amount_due, then [15]
        the clerk gives sum - amount_due as change to the customer. [15]
```

5.3 The Only Ambiguity

The ABCVPS just happens not to have any example of the only ambiguity. However, its first sentence, The maximal number of tapes that a customer can have outstanding on rental is 20., could easily have been written using the word only, and most likely, the sentence would have been, A customer may only have 20 tapes outstanding on rental.. An informal survey of geographically close colleagues of the authors confirmed that the given sentence is indeed the common only restatement of the original sentence. However, this only sentence is wrong, in that it does not say what the sentence of which it is a translation says. The only sentence should be: A customer may have only 20 tapes outstanding on rental.. The mistaken only sentence says that the ony thing a customer may do with 20 tapes outstanding on rental is to have them, and certainly, the customer may not eat, smoke, burn, copy, or even *play* the 20 tapes outstanding on rental, unless it can be proved that these activities are part of the act of having.

The reason the sentence would most likely have been written A customer may only have 20 tapes outstanding on rental. is that the convention in English today is to put the only immediately preceding the main verb of the sentence, which is, in this case, have, regardless of where it should be put. The correct place to put the only is immediately preceding the word or phrase that is limited by the only, which is, in this case, 20 tapes. Interestingly, this convention of misplaced only seems to be only in English; in each other languages known to any of us, the word or phrase for only is placed before the word or phrase limited by the word or phrase for only. In English, words other than only suffer this misplacement problem. These other words include almost, also, even, hardly, just, merely, nearly, and really. Each of these words is a member of a class are called *limiting words*. If the ABCVPS had any of these limiting words, the word would probably have been misplaced in any sentence containing it, and the sentence could have been the example of this subsection. The lack of one of these words in the ABCVPS notwithstanding, this misplaced word problem, particularly with the words only and also, occurs frequently in NL RS as well as in most technical papers[6].

5.4 Additional Proposals for T2

The ideal T2 would be one that does all the processing described in Subsections 5.1, 5.2, and 5.3. This processing includes recognition of all instances in a RS of all the problems described in Subsections 5.1 and 5.3 followed by the rewriting described in Section 5.2. However, because those transformations require *deep* understanding of the RS text that only human beings have been able to master, we would have to settle for a less powerful tool.

The next to ideal new tool would be one that at least identified all instances in a RS of all the problems described in Subsections 5.1 and 5.3 so that a human being would not have to search for them in a large RS. The human being would not risk missing any either because she did not know of some of the classes of problems or she just missed a few due to tiredness or boredom. However, even just identifying instances of some of the problems is too complex for software. The recognition of instances of some of the problems requires uniformly correct syntactic parsing of sentences accompanied by uniformly correct identification of the parts of speech of all words of sentences. The recognition of instances of other problems requires understanding the meaning of sentences. Both requirements are beyond the capabilities of software at least today and possibly fundamentally.

A practical tool will have to recognize what it can, perhaps in creative ways totally divorced from traditional lexical, syntactic, and semantic processing, perhaps making use of statistics or of simple pattern matching at the string level. A very successful recognizer of abstractions in NL text was built by applying signal processing algorithms to whole sentences, each treated as one long stream of characters with the blank not treated different from any other character [54], in contrast to a traditional indexing program that breaks sentences into words before doing any further processing.

A key property of whatever processing the tool does concerning a particular problem p is that it have total recall of instance of p and not too much imprecision about instances of p. That is, the tool must find every instance of p in any input. It is acceptable that the tool report false positive instances of p, that are not really instances of p, so long as the number of these false positive instances of p does not overwhelm the user. If tool either fails to find at least one instance of p or inundates the user with false positives, the user might as well do the search for p manually. In the first case, the user cannot trust the tool to find every instance of p and she must look herself. In the second case, the user spends more time discarding false positives than she would spend searching for instances of p manually.

Below is a list of indicators of ambiguity and other problems that we believe may be feasible for a linguistics based tool to search for with close to 100% recall and not too much imprecision. They are listed in what we believe is increasing difficulty to achieve 100% recall with not too much imprecision. The first items involve searching

[6] This last sentence notwithstanding, this misplaced word problem does *not* occur in this paper. The authors made sure of that!

for specific words or specific lexical patterns. The last items involve parsing, part of speech identification, and referent identification. The extra space between some pairs of items serves to group together items of similar difficulty requiring similar processing. The citations after each item give least one source of more information about the item.

- slash, especially and/or [12, 14]
- potentially nonparenthetical parentheses [12, 14]
- respectively [12, 14]
- potentially undefined acronym [3]

- one of specific weak words, e.g., appropriate [3, 49]
- one of specific vagueness-revealing words, e.g. clearly [3, 49]
- one of specific subjectivity-revealing words, e.g. similar [3, 49]
- one of specific optionality-revealing words, e.g. possibly [3, 49]

- demonstrative pronoun used as a noun, e.g., this is ... [48, 12, 14]
- potentially misplaced limiting word [12, 48, 14]
- potentially incorrect universal quantifier [50, 51]

- verbs joined by conjunction [50, 3]
- verb complements joined by conjunction [50, 3]
- subjects joined by conjunction [50, 3]

- unclear quantifier scope [12, 14]
- unclear anaphora [12, 3]
- unclear coordination of conjunctions [12, 14]
- negation of causality [12, 14]

- number error between anaphor and referrent [12, 14]
- unclear plural sentence [33, 46]
- presence of passive voice [50]

The rest of this subsection shows first an exploration of one of these problems, namely the only ambiguity, to see what would be involved in searching for instances of the problem in a RS and in assisting the user to understand the particulars of each instance. Then, the section describes a proposal for tool assistance to find noun synonyms in a RS.

Recall that the only ambiguity problem is that in English the convention has arisen that the word only should be put immediately before the main verb of a sentence no matter which word of the sentence is actually limited by the only. The proposal was to build a tool that would detect any sentence in which the word only appears immediately before the main verb of the sentence. We decided to ignore each sentence containing only in any place other than immediately before the main verb, because in such a sentence it is quite likely that the only is where it should be, since the user had to think about putting it in a nonconventional place.

Detecting a problematic positioning of only requires being able to, for each sentence, accurately

1. parse the sentence,
2. assign a part of speech to each word in the sentence, and
3. find the main verb of the sentence.

Then, it is a simple matter to see if the word immediately preceding the main verb is only. Given the capabilities of the year 2002 LOLITA-based tool $T1_1$ described in Sections 4.1–4.3, we have every reason to hope that this detection can be done quickly and accurately with the even more advanced parsers available now, such as that built by Sleator and Temperly (S&T) at Carnegie Mellon University [34]. At least one group has reported considerable satisfaction with the S&T parser [55].

We explored several proposals for what the tool would report to the user to help the user understand the potential ambiguity in each detected instance of the only problem. The two extremes are

1. to ask the user if she really meant what she said by showing her what she really said and then suggesting an alternative, and
2. to show the user all the other possible variations of her sentence.

In between lie ways of only asking the user if she really meant what she actually said.

To make these options concrete, consider the sentence,

I only nap after lunch.

Did the writer mean what she wrote, that the only thing she does after lunch is to nap? Did she mean instead,

I nap only after lunch.,

which means that the only time she naps is after lunch? Another possible, although less likely, meaning, is that of

I nap after only lunch.,

which means that she naps only after lunch and not after any other event. There is one other, not so likely meaning, that of

Only I nap after lunch.,

which means that among all the people under consideration in the conversation, only she, i.e., the "I" in the sentence, naps after lunch. In fact, statistically, the most probable correct sentence is the second, I nap only after lunch.

One way of the tool's showing the user what she really said and then offering an alternative would be for the tool to say:

You said "*I only nap after lunch*."

Do you really mean that the only action *I* does *after lunch* is to *nap*?

Perhaps, you mean to say "*I nap* only *after lunch*."

Each italicized portion is a piece that the user has written, and each piece is plugged into appropriate holes in a template for a question and a proposed alternative. The template is

You said “[*Full sentence*.]”

Do you really mean to say that the only action [*Subject*] does [*Complement beginning with preposition*] is to [*Verb in present tense*]?

Perhaps, you mean to say “[*Subject*] [*Verb in present tense*] only [*Complement beginning with preposition*].”

The template for the offered alternative constructs the alternative that is statistically the most likely intent of the writer.

The idea of the proposed response is that once a user has been shown by the question what the sentence she wrote really means and she has been offered what is probably what she meant to write, she will be able to either to easily correct her sentence or to insist with confidence that what she wrote *was* correct.

In order for a tool to build this sort of response from an input sentence, the tool must be able

1. to parse the sentence,
2. to determine where and what the subject, verb, and complement are,
3. to determine that the verb is in present tense, and
4. to determine that the complement begins with a preposition.

With all this information determined, the tool can select the template for an only preceding a present tense main verb that has a complement beginning with a preposition.

A sentence with an only preceding a past tense main verb that has a complement that is a singular direct object, e.g.,

He only brought lunch.

would get the response

You said “*He only brought lunch*.”

Do you really mean to say that the only action *He* did to *lunch* was to have *brought* it?

Perhaps, you mean to say “*He brought* only *lunch*.”

Again, observe that a correct parse is necessary to construct the response to the input sentence. When we experimented with a parser to see if we could extract enough information from the parser to recognize

I only nap after lunch.

as an instance of a potentially misplaced only and to fill in the correct template correctly, we were disappointed immediately. The parser we used, whose name shall forever remain anonymous to protect the guilty, failed to even find a verb in the sentence; it had classified nap as a noun!

A proposal between the extremes is to use a question template that requires identifying only the main verb of the sentence. Note that the main verb must already have been identified in order to have determined that the only immediately precedes the main verb. So for the sentence, I only nap after lunch.,

the tool would respond:

You said, "*I only nap after lunch.*"

Do you really mean to say that only action the subject does is *nap*?

The other example,

He only brought lunch.,

would get the response

You said "*He only brought lunch*."

Do you really mean to say that the only action the subject does is *brought*?

Perhaps, these questions are not as illuminating to the user as the previous set that requires more information from the parse of a sentence. However, after the first few times facing this sort of question, a reasonably intelligent user will learn the meaning of the question and will be able to respond correctly to the tool, and more importantly, to fix her own sentence if it is not correct.

A third proposal, in the other extreme, requires identifying that an only appears before a word that only *may* be the main verb of the sentence. For the sentence,

I only nap after lunch.,

the tool would respond:

You said, "I only nap after lunch."

Did you mean that or

"Only I nap after lunch." or

"I nap only after lunch." or

"I nap after only lunch."?

Note that tool does not have to really identify anything correctly other than the word only. If the user had written

I nap only after lunch.,

and the parser had somehow, but incorrectly, determined that after is the main verb, the tool would output:

You said, "I nap only after lunch."

Did you mean that or

"Only I nap after lunch." or

"I only nap after lunch." or

"I nap after only lunch."?

While such an output from the tool would probably be a waste of time, it is not incorrect in the sense that it does correctly say what the user said and it does offer bona fide alternatives.

The only reason that we suggest that the tool at least try to identify that the only appears before the main verb of the sentence is to reduce the incidence of unnecessary questioning of the input so as not to inundate the user.

In fact, it may very well be that it is worth abandoning parsing entirely and searching for only only on the grounds that most people almost always put their onlys in the wrong places and put their onlys in the right places by accident. This simple approach would work for any problem that can be identified by the presence of a keyword.

The issue is to find the right balance between the recall and precision of the underlying parser and the recall and precision of the tool. For the first kinds of responses, the more illuminating and more precise are the response of the tool the more recall is required of the parser. No parser has total recall, so the responses will not have total recall. For the second kind of response, total recall is possible, but the issue will be how much imprecision the user is burdened with. The less precise is the recognition of the verb, the more unnecessary and useless questions the user will be asked.

The final idea offered in this subsection is a way to help coalesce a set of synonym noun into one representative term. The idea is based on the recognition that really only a human being can accurately identify synonyms. After the tool has somehow identified all words that it considers nouns and all phrases that it considers noun phrases, it should form one list of nouns and noun phrases, perhaps in alphabetical order. It should then present to the user each pair of items from the list and ask for a quick “Y”, “N”, or “?” response to each pair indicating that the elements of the pair are synonyms, that they are not synonyms, or that the user does not know enough to tell, respectively. The factors in the tradeoff are:

– how accurately the tool can distinguish nouns; the tool may not even have to parse to determine nouns from rules and a list of known nouns,
– how much inaccurate identification of nouns affects the effectiveness of the process, that is
  • if the recognition of nouns achieves more precision at the cost of recall, will the tool miss too many pairs for the user’s confidence?
  • if the recognition of nouns achieves more recall at the cost of precision, will the tool inundate the user with too many useless pairs?
– how many pairs of nouns and noun phrases are there in a document compared to the speed of the user’s replying to the tool; the number of nouns in a document does not grow even linearly with the length of a document, because adding sentences about the same subject does not necessarily add new nouns.

6 Conclusions

This paper describes a two-step, tool-assisted approach to identifying ambiguities in NL RSs. In the first step, T1 would be used to apply a set of lexical and syntactice ambiguity measures to a RS in order to identify potentially ambiguous sentences in the RS. In the second step, T2 would show what specifically is potentially ambiguous about each sentence in the RS. The paper describes the use of a shell-script and a manual mock-up prototype for T1 and T2 for the purpose of exploring their requirements. Experimental application of the prototypes to several small RSs has shed some light on the requirements for T1 and T2. More work is needed, and it is being done now.

6.1 Settled Requirements Issues

The requirements issues settled by the experiments include:

– $T1_1$ built using the NL parser of LOLITA is effective in calculating the syntactic ambiguity of sentences, but it is too expensive both by its use of a heavy weight NLP tool and a NLP tool that has gone commercial. However, other, publically accessible parsers should work equally well as the underlying NL parser.
– WordNet is no worse than other lexical resources in calculating the functions of UM that depend on lexical ambiguity, α and γ but provides additional capabilities that may prove useful.
– In calculating the lexical ambiguity of a sentence, γ, SA_{sum} is as effective as the more complex $SA_{logProd}$ but is cheaper to calculate.
– Color coding is an effective way to indicate degrees of ambiguity of individual sentences.
– $T1_4$, built out of publically accessible resources, is effective in calculating the lexical ambiguity of sentences using WordNet as its dictionary and SA_{sum} as γ's auxiliary function.
– There are a number of indicators of semantic, programatic, software-engineering, and language-error ambiguities that are feasible to search for in NL RSs, such that merely reporting them provides useful information to a RS analyst.
– A combination of lexical and syntactic methods can be used to find these semantic, programatic, software-engineering, and language-error ambiguity indicators.

6.2 Counter Indications

Each tool was tried on one or more small NL RSs to see which of each RS's sentences it would classify as ambiguous, either highly, hardly, or somewhat ambiguous. $T1_1$ was successful in identifying sentences with syntactic ambiguity, i.e., sentences with multiple parses, $T1_4$ was successful in identifying sentences with lexical ambiguity, i.e., containing words with multiple meanings.

However, when the author Berry, a native English speaker looked at the same NL RSs, his conclusions were that each looked okay. In particular, none of the lexical ambiguity seemed to matter, because it disappeared when a whole NL RS was considered as a single context that pins down the meaning of each supposedly ambiguous word. As for syntactic ambiguity, it is possible that there were syntactic ambiguities that he did not notice because he unbconsciously disambiguated [15] each sentence to its intended meaning.

Berry did see serious ambiguity problems not even considered by $T1_1$ and $T1_4$. This observation led to the research into the requirements for T2. Section 5.1 details all the difficulties caused by all the inappropriate definite articles, passive voice, plural subject, and synonyms appearing in the NL RS that was subjected to both $T1_1$ and $T1_4$. Section 5.2 shows a rewrite of this NL RS that eliminates these particular ambiguities. The facts that (1) these serious problems are not among those measured by $T1_1$ and $T1_4$ and that (2) the problems measured by $T1_1$ and $T1_4$ did not seem to be relevant raise some question about the usefulness of NLP-based tools that focus on measuring lexical and syntactic ambiguity. It seems that the kinds of ambiguity important to requirements analysis are semantic, pragmatic, software engineering, and language error. On the other hand, the lexical and syntactic information extracted while computing the lexical and syntactic ambiguity measures are essential for detecting the indicators of semantic, pragmatic, software-engineering, and language-error ambiguities. This caution recalls the warning issued a long time ago by Kevin Ryan that applying NLP tools to RE problems was fraught with difficulties [56].

Adding to this caution are the conclusions from experiments performed by the first three authors of this paper plus John Mylopoulos [57] to test the effectiveness and efficiency of NL-OOPS, a LOLITA based tool for constructing an object-oriented domain model, a.k.a., conceptual model, from a NL RS. Indeed, the experiments used the same Softcom and Library problem statements used in Section 7 of this paper. The expriments compared the quality of domain class models produced by teams and individuals working with NL-OOPS from the Softcom and Library problem statements to the quality of domain class models produced by teams and individuals working manually from the same problems statements. The empirical results from three experiments neither confirm nor refute the hypotheses that the quality of a domain class model is higher if its development is supported by a NLP system [57].

One particular problem that reduced the quality of the models produced by tool-assisted groups was an *inertial effect*. Even though the tool-assisted groups worked faster than the other groups, the tool-assisted groups' models were not as good as those of the other groups. The tool-assisted groups' models had more unnecessary classes, which had been suggested by the tool, and fewer essential classes, which had been missed by the tool, than did the manually working groups' models. It appears that the tool took away some incentive to think the problem through thoroughly. A tool-using group could see that the tool was doing a lot of thinking in presented a list of suggested classes. A manually working group had no choice but to think from the beginning, and the thinking appears to have resulted in a more complete but economical model.

Compare these less than stellar results with those of Goldin, who had a similar goal, to build a tool, AbstFinder, to help identify abstractions in NL text [54]. AbstFinder uses a signal processing algorithm to find sufficiently often appearing noncontiguous snippets of text each of whose pieces may be less than a full word. Each such snippet is presented to the user as a potential abstraction, and the user must decide if it is indeed an abstraction. In Goldin's experiment, one AbstFinder user, Goldin herself, found in 8 hours of work more abstractions in an industrial RFP than a team of three domain experts working for one month.

There is considerable difference between NL-OOPS and AbstFinder in apparent polish of the output and in apparent intelligence. NL-OOPs presents polished lists of actual class names and appears more intelligent than AbstFinder which presents unpolished snippets of text of potential abstractions that have to be given names by the user. The user of AbstFinder has to be more engaged just to use the tool properly. Perhaps the difference of the tools in the engagement of their users leads to differences in the thoroughness of the thinking about their outputs. This difference in engagement could account for the observations about the quality of the models of the tool-assisted teams.

These observations lead to hypotheses about applying NLP tools to NLRSAI&M.

- Any tool claiming to help a person find particular kinds of ambiguities in a RS must guarantee 100% recall of instances in the RS of these kinds of ambiguities. Otherwise, the user will not use the tool because she must go through the RS manually anyway to find the instances that the tool missed. These missed instances might be harder to find than otherwise because they have become the proverbial needles in a haystack.
- For any kind of ambiguity in a RS, there is a tradeoff between recall and precision that can be exercised by the choice of search algorithm. For example, searching for only those onlys that appear before verbs cannot achieve 100% recall because there is no algorithm that is 100% accurate in parsing and part-of-speech identification. Just reporting every instance of only achieves 100% recall of onlys that appear before verbs at the cost of low precision in that many false positives, i.e., onlys that do not appear before verbs, are reported.
- There is a fine line between engaging a user and encouraging an inertial effect. Too much polish in the output, perhaps even too much precision in the output disengages the user and encourages an inertial effect of accepting the output without question. A little bit of imprecision engages the user by forcing her to think about whether a given instance of a potential ambiguity is indeed an ambiguity.
- There is a fine line between engaging and inundating a user. Just enough imprecision in the output gets the user to think about the output. Too much imprecision results in the user spending a lot of time thinking about an rejecting false positives to the point that she perceives that she will finish faster by working manually, even though she risks missing some trued positives.

Thus any tool for NLRSAI&M should have

- 100% recall,
- some, but not too much imprecision, and
- high summarization, i.e., the size of the output that the user must wade through is a small fraction of the size of the input to the tool.

Anyone building a tool to assist in NLRSAI&M must pay attention to these tradeoffs. We believe that these tradeoffs apply to *any* tool that purports to help any process in software engineering.

Bibliography

[1] 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-97), New York, NY, USA, ACM Press (1997) 161–171

[2] Mich, L., Garigliano, R.: Ambiguity measures in requirement engineering. In Feng, Y., Notkin, D., Gaudel, M., eds.: Proceedings of International Conference on Software—Theory and Practice (ICS2000), Sixteenth IFIP World Computer Congress, Beijing, Publishing House of Electronics Industry (2000) 39–48

[3] 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. (2001) 97–105

[4] Mich, L.: On the use of ambiguity measures in requirements analysis. In Moreno, A., van de Riet, R., eds.: Proceedings of the Sixth International Conference on Applications of Natural Language to Information Systems (NLDB). (2001) 143–152

[5] Mich, L., Franch, M., Inverardi, P.N.: Requirements analysis using linguistic tools: Results of an on-line survey. Requirements Engineering Journal **9** (2004) 40–56

[6] Garigliano, R., Nettleton, D.J.: Neo-pragmatism. In Group, T.L., ed.: The LOLITA Project: the First Ten Years, Vol. 3. Springer Verlag (1997)

[7] Morgan, R., Garigliano, R., Callaghan, P., Poria, S., Smith, M., Urbanowicz, A., Collingham, R., Costantino, M., Cooper, C.: Description of the LOLITA system as used in MUC-6. In: Proceedings of the Sixth Message Understanding Conference (MUC-6. (1995)

[8] Garigliano, R., Urbanowicz, A., Nettleton, D.J.: Description of the LOLITA system as used in MUC-7. In: Proceedings of the Message Understanding Conference (MUC-7). (1998) `http://acl.ldc.upenn.edu/muc7/`.

[9] Mich, L., Garigliano, R.: NL-OOPS: A requirements analysis tool based on natural language processing. In: Proceedings of Third International Conference on Data Mining. (2002) 321–330

[10] Grishman, R., Sundheim, B.: Design of the MUC-6 evaluation. In: Proceedings of the Sixth Message Understanding Conference (MUC-6), San Francisco, CA, USA, Morgan Kaufmann (1995) 1–11

[11] Kiyavitskaya, N., Zeni, N., Mich, L., Berry, D.M.: Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. Technical report, School of Computer Science, University of Waterloo, Waterloo, ON, Canada (2007) `http://se.uwaterloo.ca/˜dberry/FTP_SITE/tech.reports/KZMB2007AmbTR.pdf`.

[12] Berry, D.M., Kamsties, E., Krieger, M.M.: From contract drafting to software specification: Linguistic sources of ambiguity. Technical report, University of Waterloo (2003) `http://se.uwaterloo.ca/˜dberry/handbook/ambiguityHandbook.pdf`.

[13] Berry, D.M., Kamsties, E.: Ambiguity in requirements specification. In Leite, J., Doorn, J., eds.: Perspectives on Requirements Engineering. Kluwer, Boston, MA, USA (2004) 7–44

[14] Berry, D.M., Bucchiarone, A., Gnesi, S., Lami, G., Trentanni, G.: A new quality model for natural language requirements specifications. In: Proceedings of the International Workshop on Requirements Engineering: Foundation of Software Quality (REFSQ). (2006)

[15] Gause, D.C.: User DRIVEN Design—The Luxury that has Become a Necessity, A Workshop in Full Life-Cycle Requirements Management. ICRE 2000 Tutorial T7, Schaumberg, IL, USA (2000)

[16] Lyons, J.: Semantics I and II. Cambridge University Press, Cambridge, UK (1977)

[17] Hirst, G.: Semantic Interpretation and the Resolution of Ambiguity. Studies in Natural Language Processing. Cambridge University Press, Cambridge, UK (1987)

[18] Allen, J.: Natural Language Understanding. Second edn. Addison-Wesley, Reading, MA, USA (1995)

[19] Levinson, S.: Pragmatics. Cambridge University Press, Cambridge, UK (1983)

[20] Walton, D.: Fallacies Arising from Ambiguity. Applied Logic Series. Kluwer Academic, Dordrecht, NL (1996)

[21] Ide, N., Véronis, J.: Word sense disambiguation: The state of the art. Computational Linguistics **24** (1998) 1–40

[22] Yarowsky, D.: Word-sense disambiguation using statistical models of roget's categories trained on large corpora. In: Proceedings of COLING-92. (1992) 454–460

[23] ACL-SIGLEX: Sens Eval Web Site. University of North Texas, Denton, TX, USA (accessed 12 March 2006) `http://www.senseval.org/`.

[24] Harper, K.E.: Semantic ambiguity. Mechanical Translation **4** (1957) 68–69

[25] Harper, K.E.: Contextual analysis. Mechanical Translation **4** (1957) 70–75

[26] Mitamura, T.: Controlled language for multilingual machine translation. In: Proceedings of Machine Translation Summit VII. (1999)

[27] Chantree, F.: Ambiguity management in natural language generation. In: Seventh Annual CLUK Research Colloquium. (2004)

[28] Kamsties, E., Berry, D., Paech, B.: Detecting ambiguities in requirements documents using inspections. In Lawford, M., Parnas, D.L., eds.: Proceedings of the First Workshop on Inspection in Software Engineering (WISE'01). (2001) 68–80

[29] Denger, C.: High quality requirements specifications for embedded systems through authoring rules and language patterns. Technical Report M. Sc. Thesis, Fachbereich Informatik, Universität Kaiserslautern (2002)

[30] Denger, C., Berry, D.M., Kamsties, E.: Higher quality requirements specifications through natural language patterns. In: Proceedings of the IEEE International Conference on Software-Science, Technology & Engineering (SwSTE'03), IEEE Computer Society Press (2003) 80–89

[31] Fuchs, N.E., Schwitter, R.: Specifying logic programs in controlled natural language. In: CLNLP'95, Workshop on Computational Logic for Natural language Processing. (1995)

[32] Fuchs, N.E., Schwitter, R.: Attempto controlled English. In: CLAW'96, The First International Workshop on Controlled Language Applications. (1996)

[33] Schwertel, U.: Controlling plural ambiguities in Attempto Controlled English. In: Proceedings of the Third International Workshop on Controlled Language Applications (CLAW), Seattle, WA, USA (2000)
[34] Sleator, D.D., Temperley, D.: Parsing English with a link grammar. In: Proceedings of the Third International Workshop on Parsing Technologies. (1993) `http://www.link.cs.cmu.edu/link/papers/index.html`.
[35] for Computational Linguistics, I.: TreeTagger—A Language Independent Part-of-Speech Tagger. Institute for Natural Language Processing, University of Stuttgart, Stuttgart, DE (accessed 14 March 2007) `http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/`.
[36] Koller, T.: TreeTagger Output Visualiation Module. University of Nottingham, Nottingham, UK (accessed 14 March 2007) `http://www.cele.nottingham.ac.uk/˜cczt k/treetagger.php`.
[37] Miller, G.A., Felbaum, C., *et al*: WordNet Web Site. Princeton University, Princeton, NJ, USA (accessed 12 March 2006) `http://wordnet.princeton.edu/`.
[38] Garigliano, R., Boguraev, B., Tait, J.: Editorial. Journal of Natural Language Engineering **1** (1995) 1–7
[39] Cooper, K., Ito, M.: SRRS training material. Technical Report CICSR-TR99-001, University of British Columbia (1999)
[40] Kellogg, M.: WordReference Web Site. WordReference.com, McLean, VA, USA (accessed 12 March 2006) `http://www.wordreference.com/`.
[41] Orgad, Z.: Babylon Web Site. Babylon, Ltd., Or Yehuda, Israel (accessed 12 March 2006) `www.babylon.com`.
[42] Rational Corp.: IBM Rational Rose Web Site. IBM (accessed 12 March 2006) `http://www-306.ibm.com/software/rational/`.
[43] Guiasu, S.: Information Theory with Applications. McGraw Hill, New York, NY, USA (1977)
[44] Rolland, C., Proix, C.: A natural language approach for requirements engineering. In Loucopoulos, P., ed.: CAiSE'92, Fourth International Conference on Advanced Information Systems Engineering. LNCS 593, Springer Verlag (1992) 257–277
[45] Eriksson, H.E., Penker, M.: UML Toolkit. John Wiley, New York, NY, USA (1998)
[46] Berry, D.M., Kamsties, E.: The syntactically dangerous *all* and plural in specifications. IEEE Software **22** (2005) 55–57
[47] Bucchiarone, A., Gnesi, S., Pierini, P.: Quality analysis of NL requirements: An industrial case study. In: Proceedings of the Thirteenth IEEE International Conference on Requirements Engineering (RE'05). (2005) 390–394
[48] Dupré, L.: Bugs in Writing: A Guide to Debugging Your Prose. second edn. Addison-Wesley, Reading, MA, USA (1998)
[49] Kovitz, B.L.: Practical Software Requirements: A Manual of Content and Style. Manning, Greenwich, CT, USA (1998)
[50] Rupp, C., Goetz, R.: Linguistic methods of requirements-engineering (NLP). In: Proceedings of the European Software Process Improvement Conference (EuroSPI). (2000) `http://www.iscn.com/publications/#eurospi2000`.

[51] Berry, D.M., Kamsties, E.: The dangerous ‘all’ in specifications. In: Proceedings of 10th International Workshop on Software Specification & Design, IWSSD-10, IEEE Computer Society Press (2000) 191–194

[52] Götz, R., Rupp, C.: Regelwerk natürlichsprachliche methode. Technical report, Sophist (1999) `http://www.sophist.de`.

[53] Jackson, M.A., Zave, P.: Domain descriptions. In: Proceedings of the IEEE International Symposium on Requirements Engineering, Los Alamitos, CA, USA, IEEE Computer Society Press (1993) 56–64

[54] Goldin, L., Berry, D.M.: AbstFinder: A prototype abstraction finder for natural language text for use in requirements elicitation. Automated Software Engineering **4** (1997) 375–412

[55] Popescu, D., Rugaber, S., Medvidovic, N., Berry, D.M.: Improving the quality of requirements specifications via automatically created object-oriented models. Technical report, University of Southern California (2007)

[56] Ryan, K.: The role of natural language in requirements engineering. In: Proceedings of the IEEE International Symposium on Requirements Engineering, Los Alamitos, CA, USA, IEEE Computer Society Press (1993) 240–242

[57] Kiyavitskaya, N., Zeni, N., Mich, L., Mylopoulos, J.: Experimenting with linguistic tools for conceptual modeling: Quality of the models and critical features. In Farid Meziane, E.M., ed.: Natural Language Processing and Information Systems, Proceedings of the Ninth International Conference on Applications of Natural Language to Information Systems (NLDB), Springer, LNCS 3136 (2004) 135–146

[58] IEEE: IEEE Recommended Practice for Software Requirements Specifications, ANSI/IEEE Standard 830-1993. Institute of Electrical and Electronics Engineering, New York, NY, USA (1993)

[59] Parnas, D.L.: Personal communication via electronic mail (2002)

[60] Davis, A.: Software Requirements: Objects, Functions, and States. Prentice Hall, Englewood Cliffs, NJ, USA (1993)

[61] Kamsties, E.: Understanding ambiguity in requirements engineering. In Aurum, A., Wohlin, C., eds.: Engineering and Managing Software Requirements. Springer, Berlin, Germany (2005) 245–266

[62] Gunter, C.A., Gunter, E.L., Jackson, M.A., Zave, P.: A reference model for requirements and specifications. IEEE Software **17** (2000) 37–43

[63] Jarke, M., Rolland, C., Sutcliffe, A., Dömges, R.: The NATURE of Requirements Engineering. Shaker Verlag, Aachen, Germany (1999)

[64] Parnas, D.L., Asmis, G.J.K., Madey, J.: Assessment of safety-critical software in nuclear power plants. Nuclear Safety **32** (1991) 189–198

Appendix 1: Experiment Data

Table 9 shows for each menu item word, for each dictionary, the number of senses and

	WordNet		WordReference		Babylon		Weighted
Word	Senses	Roles	Senses	Roles	Senses	Roles	Average
File	9	2	16	2	9	2	11.7
New	12	2	18	2	9	2	13.4
Open	15	2	55	3	27	3	34.1
Save	11	2	12	4	8	3	10.5
Autosave	NP	NP	NP	NP	1	1	0.3
As	3	2	27	5	3	3	12.3
Log	8	2	13	1	10	2	10.5
Clear	46	4	52	4	18	4	39.9
Load	12	2	28	2	12	2	18.2
Model	16	3	15	2	9	3	13.5
Workspace	1	1	NP	NP	1	1	0.6
Units	6	1	14	1	5	1	8.8
Unload	2	1	6	1	2	1	3.6
Control	19	2	14	2	8	2	13.8
Uncontrol	NP	NP	NP	NP	NP	NP	0
Write	9	1	17	1	5	1	10.9
Protection	7	1	6	1	2	1	5.1
Import	7	2	7	2	7	2	7.0
Export	3	2	3	2	3	2	3.0
Update	4	2	2	2	2	2	2.6
Print	10	2	17	2	10	2	12.7
Page	9	2	15	2	7	2	10.7
Setup	3	1	16	3	3	1	8.1
Edit	4	1	6	2	2	1	4.2
Path	4	1	4	1	3	1	3.7
Map	8	2	9	2	5	2	7.5
Exit	6	2	11	2	2	2	6.7
Undo	5	1	4	1	5	2	4.6
Redo	2	1	2	1	1	1	1.7
Cut	73	3	80	3	16	3	58.6
Copy	8	2	9	2	4	2	7.2
Active	19	2	12	2	4	2	11.8
Diagram	2	2	3	2	3	2	2.7
Paste	6	2	10	2	6	2	7.6
Delete	3	1	1	1	1	1	1.6
Select	3	2	1	1	4	2	2.5
All	3	2	25	3	8	2	13.0
From	NP	NP	7	1	4	1	3.9

Table 9. Number of Senses and Syntactic Roles for Menu Item Words

the number of syntactic roles found in the word's entry in the dictionary. The weighted average column gives for each word the average number of senses found per dictionary, with the value for a dictionary weighted by the dictionary's dimension. If a word is simply not present in a dictionary, then its entry for that dictionary shows "NP" ("not present").

Table 10 shows the $SA_{logProd}$ value for each sentence of the Library Problem

Sentence Index	WordReference	Babylon	WordNet
1	27.6	39.8	17.9
2	13.1	28.6	11.8
3	20.5	34.5	16.3
4	19.8	41.6	18.5
5	16.7	23.5	16.0
6	35.6	50.7	32.3
7	23.7	58.9	28.2
8	33.2	62.0	24.2
9	24.3	51.6	28.1
10	33.0	63.8	28.0
11	18.6	34.6	14.5
12	12.5	16.3	9.2
13	13.1	40.6	12.1
14	26.1	45.6	27.6
15	14.5	22.4	12.0
16	18.8	31.2	18.7
Average	21.9	40.4	19.7

Table 10. $SA_{logProd}$ Value for Each Sentence of Library Problem Statement

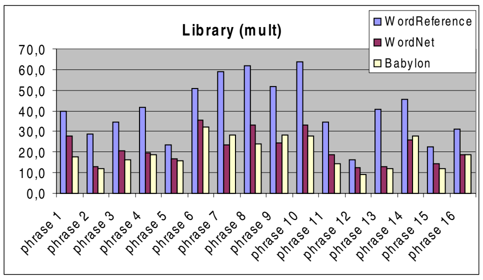


Fig. 10. Plot of $SA_{logProd}$ Value for Each Sentence of Library Problem Statement

Statement according to each dictionary. The last row gives the average $SA_{logProd}$ value over all the sentences in the problem statement. Figure 10 shows a plot of these sentence data.

Table 11 shows the $SA_{logProd}$ value for each sentence of the Softcom Problem

Sentence Index	WordNet	WordReference	Babylon
1	32.2	57.4	41.5
2	18.6	30.9	21.7
3	15.7	25.2	11.9
4	19.1	32.4	20.7
5	9.8	26.8	14.2
6	22.2	37.5	26.3
7	26.8	36.9	22.6
8	8.2	12.1	6.2
9	35.8	57.5	35.5
10	30.4	49.9	27.2
11	16.8	25.0	13.7
12	16.9	27.8	20.8
13	12.2	19.1	9.7
14	19.3	34.7	17.5
15	18.7	37.2	21.8
16	13.1	27.5	15.5
17	30.9	43.0	24.3
18	24.1	32.9	18.4
Average	20.6	34.1	20.5

Table 11. $SA_{logProd}$ Value for Each Sentence of Softcom Problem Statement

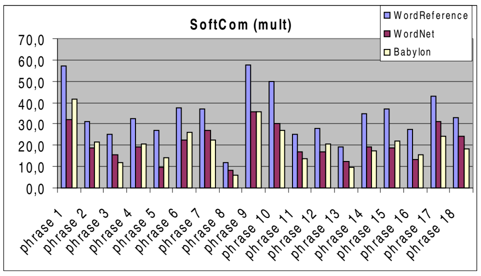


Fig. 11. Plot of $SA_{logProd}$ Value for Each Sentence of Softcom Problem Statement

Statement according to each dictionary. The last row gives the average $SA_{logProd}$ value over all the sentences in the problem statement. Figure 11 shows a plot of these sentence data. Note that sentence i is called "phrase i", because the Italian word for "sentence" is "frase".

Table 12 shows the SA_{sum} value for each sentence of the Library Problem State-

Sentence Index	WordNet	WordReference	Babylon
1	177	45	102
2	119	29	38
3	144	46	72
4	174	49	57
5	85	44	50
6	200	87	113
7	222	84	114
8	289	68	99
9	201	72	92
10	260	87	128
11	128	35	52
12	48	21	30
13	134	32	41
14	244	75	108
15	102	28	44
16	115	49	75
Average	165.1	53.2	75.9

Table 12. SA_{sum} Value for Each Sentence of Library Problem Statement

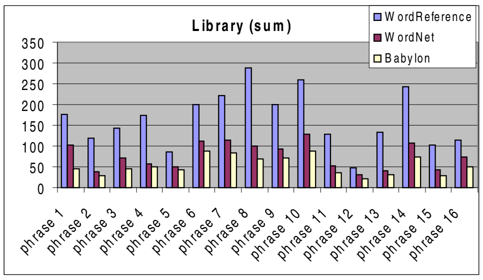


Fig. 12. Plot of SA_{sum} Value for Each Sentence of Library Problem Statement

ment according to each dictionary. The last row gives the average SA_{sum} value over all the sentences in the problem statement. Figure 12 shows a plot of these sentence data.

Table 13 shows the SA_{sum} value for each sentence of the Softcom Problem State-

Sentence Index	WordReference	Babylon	WordNet
1	246	115	100
2	214	78	100
3	171	37	59
4	218	75	95
5	75	33	24
6	142	71	62
7	178	67	109
8	38	14	23
9	254	116	125
10	242	73	123
11	107	35	43
12	107	66	51
13	76	27	29
14	181	51	61
15	161	61	63
16	111	47	45
17	184	70	110
18	158	50	92
Average	159.1	60.3	73.0

Table 13. SA_{sum} Value for Each Sentence of Softcom Problem Statement

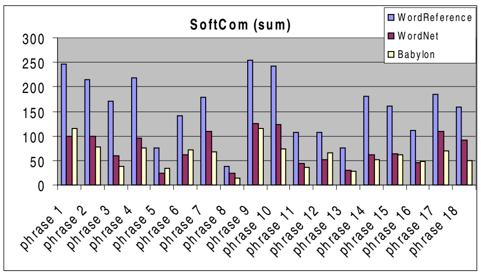


Fig. 13. Plot of SA_{sum} Value for Each Sentence of Softcom Problem Statement

ment according to each dictionary. The last row gives the average SA_{sum} value over all the sentences in the problem statement. Figure 13 shows a plot of these sentence data.

Table 14 shows the average and weighted average $SA_{logProd}$ values for each sen-

Sentence Index	Average	Weighted Average
1	28.4	29.4
2	17.8	18.7
3	23.8	24.7
4	26.6	27.8
5	18.7	19.1
6	39.5	40.5
7	36.9	38.7
8	39.8	41.7
9	34.7	36.1
10	41.6	43.5
11	22.5	23.6
12	12.7	13.0
13	21.9	23.5
14	33.1	34.1
15	16.3	16.8
16	22.9	23.6
Averages	27.3	28.4

Table 14. Average and Weighted Average $SA_{logProd}$ Values for Each Sentence of Library Problem Statement

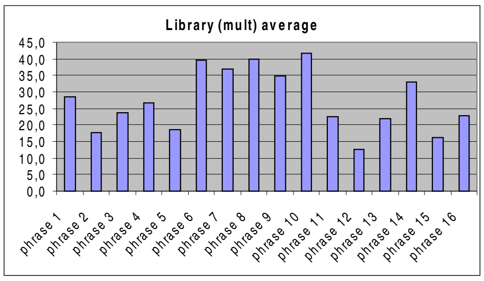


Fig. 14. Plot of Average $SA_{logProd}$ Value for Each Sentence of Library Problem Statement

tence of the Library Problem Statement according to the three dictionaries. The last row gives the averages of the average and weighted average $SA_{logProd}$ values over all the sentences in the problem statement. Figure 14 and Figure 15 show plots of these sentence average and weighted average data.

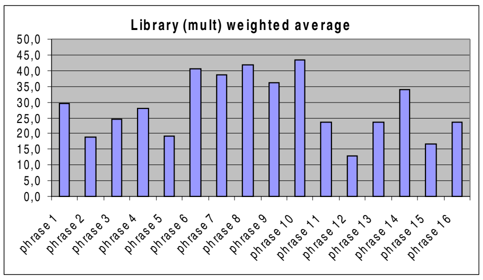


Fig. 15. Plot of Weighted Average $SA_{logProd}$ Value for Each Sentence of Library Problem Statement

Table 15 shows the average and weighted average $SA_{logProd}$ values for each sen-

Sentence Index	Average	Weighted Average
1	43.7	44.8
2	23.8	24.3
3	17.6	18.2
4	24.1	24.7
5	16.9	17.7
6	28.7	29.4
7	28.8	29.5
8	8.8	9.1
9	42.9	44.2
10	35.8	37.0
11	18.5	19.1
12	21.8	22.3
13	13.7	14.1
14	23.8	24.7
15	25.9	26.8
16	18.7	19.4
17	32.7	33.6
18	25.1	25.8
Averages	25.1	25.8

Table 15. Average and Weighted Average $SA_{logProd}$ Values for Each Sentence of Softcom Problem Statement

tence of the Softcom Problem Statement according to the three dictionaries. The last row gives the averages of the average and weighted average $SA_{logProd}$ values over all the sentences in the problem statement. Figure 16 and Figure 17 show plots of these sentence average and weighted average data.

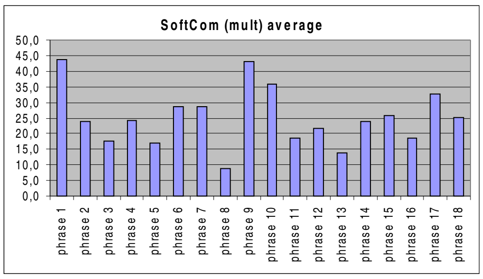


Fig. 16. Plot of Average $SA_{logProd}$ Value for Each Sentence of Softcom Problem Statement

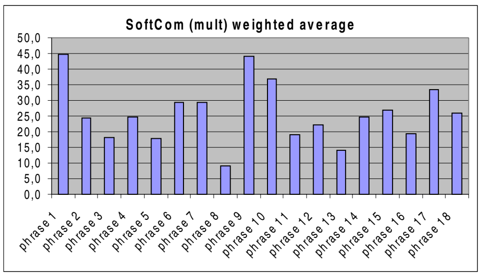


Fig. 17. Plot of Weighted Average $SA_{logProd}$ Value for Each Sentence of Softcom Problem Statement

Table 16 shows the average and weighted average SA_{sum} values for each sentence

Sentence Index	Average	Weighted Average
1	108	114.1
2	62	66.8
3	87.3	92.2
4	93.3	100.1
5	59.7	61.8
6	133.3	139.0
7	140.0	147.0
8	152	163.6
9	121.7	128.4
10	158.3	167.0
11	71.7	76.4
12	33.0	34.3
13	69.0	74.4
14	142.3	151.0
15	58.0	61.7
16	79.7	82.8
Averages	98.1	103.8

Table 16. Average and Weighted Average SA_{sum} Values for Each Sentence of Library Problem Statement

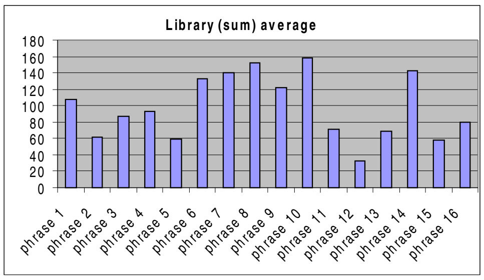


Fig. 18. Plot of Average SA_{sum} Value for Each Sentence of Library Problem Statement

of the Library Problem Statement according to the three dictionaries. The last row gives the averages of the average and weighted average SA_{sum} value over all the sentences in the problem statement. Figure 18 and Figure 19 show plots of these sentence average and weighted average data.

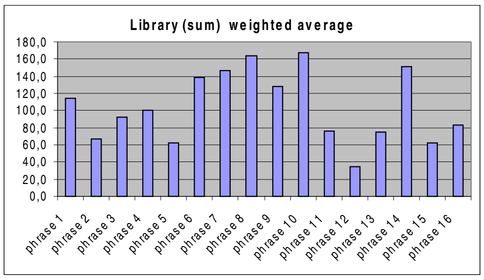


Fig. 19. Plot of Weighted Average SA_{sum} Value for Each Sentence of Library Problem Statement

Table 17 shows the average and weighted average SA_{sum} values for each sentence

Sentence Index	Average	Weighted Average
1	153.7	161.2
2	130.7	137.7
3	89.0	95.9
4	129.3	136.8
5	44.0	46.5
6	91.7	95.8
7	118.0	123.2
8	25.0	26.1
9	165.0	172.4
10	146.0	154.3
11	61.7	65.5
12	74.7	77.3
13	44.0	46.7
14	97.7	104.6
15	95.0	100.5
16	67.7	71.3
17	121.3	126.8
18	100.0	105.1
Averages	97.5	102.7

Table 17. Average and Weighted Average SA_{sum} Values for Each Sentence of Softcom Problem Statement

of the Softcom Problem Statement according to the three dictionaries. The last row gives the averages of the average and weighted average SA_{sum} values over all the sentences in the problem statement. Figure 20 and Figure 21 show plots of these sentence average and weighted average data.

Appendix 2: Types of Ambiguity

Ambiguity is of great importance in many areas. For instance, in art, ambiguity is essential. Many a song or poem relies on ambiguous words for artistic effect, as in the

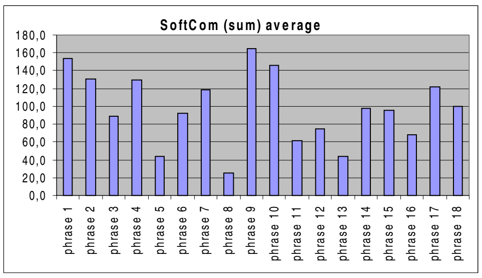


Fig. 20. Plot of Average SA_{sum} Value for Each Sentence of Softcom Problem Statement

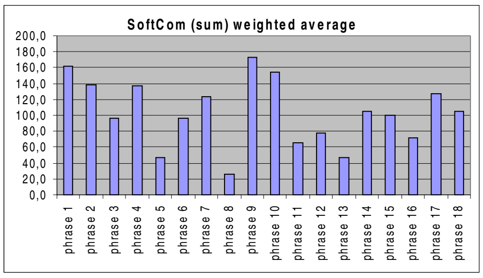


Fig. 21. Plot of Weighted Average SA_{sum} Value for Each Sentence of Softcom Problem Statement

song title Don't It Make My Brown Eyes Blue?[7], in which Blue can mean the color or sadness. In literature and rhetoric, ambiguity is used often as a source of humor. One well-known example is, Last night I shot an elephant in my pajamas. What he was doing in my pajamas I'll never know. In politics or law, on the other hand, ambiguity creates space for defining relationships or bargaining over shared goals. However, any legal document that acts as a prescription or standard for performance must be precise; accurate; consistent; and complete, in the sense of anticipating all possible contingencies. Examples of such legal documents are contracts, patents, wills, statues, political agreements, and medical prescriptions.

Another application that requires ambiguity identification is Machine Translation (MT), automatic translation from one NL to another. The existence of ambiguous words or sentences makes it difficult for an MT system to capture the meaning of a source sentence in order to produce a correct translation. However, when the source sentence is intentionally ambiguous, the ambiguity should be preserved in the translation. Therefore, any MT system must be able to identify and correctly resolve ambiguities.

Ambiguity plays an important role also in Natural Language Generation (NLG). When generating NL text, some ambiguities must be preserved and some must be eliminated, and a NLG system must be able to distinguish the two kinds of ambiguity.

Ambiguity in words must be resolved during Information Retrieval (IR) or Information Extraction (IE) to ensure that the results of a query are relevant to the intended meaning of every word in the query. Ambiguity identification is crucial also for part-of-speech tagging, speech processing, hypertext management, semantic annotation, and any other text processing application dealing with the contents of the text.

The traditional types of ambiguity include lexical, syntactic, semantic, and pragmatic ambiguity. To this list we add two additional types, software-engineering, and language-error ambiguity. [12].

Lexical Ambiguity *Lexical ambiguity* occurs when a word has several meanings. For instance, the word light as an adjective can mean "of comparatively little physical weight or density", "having relatively small amount of coloring agent", velc.[8] [37]. A word such as light, note, bear, and over, with multiple meanings, is lexically ambiguous. Lexical ambiguity can be subdivided into homonymy and polysemy. *Homonymy* occurs when two different words have the same written and phonetic representation, but unrelated meanings and different etymologies, i.e., different histories of development. Each of the homonyms has its own semantics. An example is a word bank which can mean "financial institution", "edge of a river", or "slope". *Polysemy* occurs when a word has several related meanings but one etymology. The different meanings of a polysemous expression have a base meaning in common. An example is the word point. Each of its meanings, e.g., "punctuation mark", "sharp end", "detail, argument", etc. comes from the single etymology of point.

[7] Any example text is in a sansserif typeface in order to reserve quotation marks for surrounding a quotation, the meaning of an example, and a nonexample word used as itself.

[8] "velc." is an abbreviation for "vel cetera", "or others", just as "etc." is an abbreviation for "et cetera", "and others".

Syntactic or structural ambiguity occurs when a given sequence of words can be given more than one grammatical structure, i.e. more than one parse, and each parse has a different meaning. For example, the phrase Tibetan history teacher and the sentence The police shot the rioters with guns are structurally ambiguous. The phrase Tibetan history teacher can be broken down as either (Tibetan history) teacher or Tibetan (history teacher), and the phrase The police shot the rioters with guns can be broken down as either The police shot (the rioters with guns) or The police shot (the rioters) with guns. A syntactic ambiguity can be classified as an analytical, attachment, coordination, or elliptical ambiguity.

Analytical ambiguity occurs when the role of the constituents within a phrase or sentence is ambiguous. For example, porcelain egg container can mean “a container for porcelain eggs” or “a porcelain container for eggs”.

Attachment ambiguity occurs when a particular syntactic constituent of a sentence, such as a prepositional phrase or a relative clause, can be legally attached to two parts of a sentence. A common pattern of attachment ambiguity is a prepositional phrase that may modify either a verb or a noun. For example, the sentence The girl hit the boy with a book can mean “the girl used a book to hit the boy” or “the girl hit the boy who had a book”.

Coordination ambiguity occurs when:

– more than one conjunction, and or or, is used in a sentence, e.g., I saw Peter and Paul and Mary saw me[9];
– one conjunction is used with a modifier, e.g., young man and woman.

Elliptical ambiguity occurs when it is not certain whether or not a sentence contains an ellipsis. Ellipsis is the deliberate omission of some aspect of language form whose meaning can be understood from the context of that form. Ellipsis is sometimes called gapping by linguists. An example of elliptical ambiguity is Perot knows a richer man than Trump. The sentence has two meanings. The first is that Perot knows a man who is richer than Trump is, and second is that Perot knows a man who is richer than any man Trump knows. The first meaning corresponds to the ellipsis of an implied is after Trump, and the second corresponds to the ellipsis of an implied knows after Trump.

Semantic Ambiguity *Semantic ambiguity* occurs when a sentence has more than one way of reading it within its context although it contains no lexical or structural ambiguity. Semantic ambiguity can be viewed as ambiguity with respect to the logical form, usually expressed in predicate logic, of the ambiguous sentence. Semantic ambiguity can be caused by any of:

– coordination ambiguity,
– referential ambiguity, and
– scope ambiguity.

[9] Interestingly, the Italian translation of this sentence is not ambiguous because the singular form of the translation of saw is different from the plural form of the translation of saw, and the one used depends on whether the subject of the second saw is Paul and Mary or just Mary.

Coordination ambiguity can cause both syntactic and semantic ambiguity and is as discussed under the "Lexical Ambiguity" heading.

Referential ambiguity is discussed in Section 2.1, because it is on the border line between semantic and pragmatic ambiguity. A referential ambiguity can happen within a sentence, in which case it is semantic, or between a sentence and its discourse context, in which case it is pragmatic.

Scope ambiguity occurs when a quantifier or a negation operator can enter into different scoping relations with other sentence constituents. Quantifiers include such words as every, each, all, some, several, a, etc., and negation operators include not. An example of a scope ambiguity is the sentence Every man loves a woman, which has two distinct readings: (1) "for each man there is his woman, and he loves her," and (2) "there is a single special woman who is loved by all the men". For the first reading, the scope of the universal quantifier Every contains the scope of the existential quantifier a, and for the second reading, the scope of a contains the scope of Every.

Pragmatic Ambiguity *Pragmatic ambiguity* occurs when a sentence has several meanings in the context in which it occurs. A *sentence's context* comprises its *language context*, i.e., the sentences occurring before and after the given sentence, and its *context beyond language*, i.e., the situation, the background knowledge, and the expectations of the speaker and hearer or the writer and reader of the given sentence. A pragmatic ambiguity is traditionally classified as a *referential ambiguity* or a *deictic ambiguity*.

The relation between a word or phrase and an object of the real world that the word or phrase describes is called a *reference*. An *anaphor* is an element that refers to another, preceding element, possibly in a different, but nevertheless preceding, sentence. The other, referent element is, therefore, called the anaphor's *antecedent*. Examples of anaphora include pronouns, e.g., it, they; definite noun phrases; and some forms of ellipses.

Referential ambiguity occurs when an anaphor can refer to more than one element, each playing the role of the antecedent. An example of referential ambiguity is The trucks shall treat the roads before they freeze; it is not clear what is the antecedent of they. An example of a referentially ambiguous ellipsis is in If the card is readable, then if the ATM accepts the card, the user enters the PIN. If not, the card is rejected. The ellipsis not stands for some condition X not being true; is X the ATM accepts the card or the card is readable?

Deictic ambiguity occurs when a pronoun; a time or place adverb, such as now and here; or another grammatical feature, such as tense, has more than one referent in the context outside the text. The referent can be a person in a conversation, the location the conversers are at, the current time, time, velc. In contrast to an anaphor, a deictic reference is often used to introduce a referent to the linguistic context so that it can be talked about with anaphora. An anaphor refers to something in the preceding linguistic context, but a deictic reference refers to something in the non-linguistic context. Note that a pronoun, in particular, can be anaphoric or deictic. When a pronoun refers to an element outside the preceding text, the pronoun is deictic, e.g. the you in What do you say about this idea? When the pronoun refers to an element inside the preceding text, the pronoun is anaphoric, e.g., the He in A man walked in the park. He whistled. It is possible for a given pronoun to be read as an anaphor or as deictic reference. The she in Every student thinks she is a genius. could refer to Every student, to a previously mentioned female person, or to a female other than the listener standing next to the speaker of the sentence. Thus, it is ambiguous as to whether the sentence has a scope, referential, or deictic ambiguity.

Software-Engineering Ambiguity There appears to be no single comprehensive definition of ambiguity in the software-engineering (SE) literature. Each of the following definitions highlights only some aspects of SE ambiguity and omits others. The definitions together form a complete overview of the current understanding of ambiguity in SE.

The widely used IEEE Recommended Practice for Software Requirements Specifications (SRSs) [58] says that “An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation.” Presumably, an SRS is ambiguous if it is not unambiguous.

The problem with the IEEE definition is that there is no unambiguous RS simply because for any RSs, there is always someone who understands it differently from someone else, just as there are no bug-free programs [59]. There *are* mature, usable programs whose bugs are known; the users have learned to work around the bugs and get useful computation from them. In a similar manner, there are no unambiguous RSs. Rather, there are useful specifications, each of which is understood well enough by enough people that count, enough of the implementers, a majority of the customers, and enough of the users, that it is possible to implement software meeting the specifications that does what most people expect it to do in most circumstances.

Indeed, Davis [60] has suggested a test for ambiguity: “Imagine a sentence that is extracted from an SRS, given to ten people who are asked for an interpretation. If there is more than one interpretation, then that sentence is probably ambiguous.” The problem with this test is that, as in software testing, there is no guarantee that the eleventh person will not find another interpretation. However, this test does capture the essence of a useful RS that is unambiguous for most practical purposes. Actually, we would go farther and say that the sentence *is* ambiguous, instead of just probably ambiguous. Davis provides two examples of ambiguity.

1. For up to 12 aircraft, the small display format shall be used. Otherwise, the large display format shall be used.
   Assuming that small and large display formats are defined previously, the ambiguity lies in the phrase for up to 12. Does it mean “for up to and including 12” or “for up to and excluding 12”?
2. Aircraft that are non-friendly and have an unknown mission or the potential to enter restricted airspace within 5 minutes shall raise an alert.
   Assuming again that the relevant terms are defined, the ambiguity lies in the relative precedence of and and or, because we cannot assume the precedence rules of Boolean algebra for natural language utterances.

We believe that the first interpretation of the dictionary definition, the capability of being understood in two or more possible senses or ways, is underlying Davis’s discussion of ambiguity.

A software-engineering ambiguity arises from the context that must be examined when trying to understand a sentence describing requirements [61]. As suggested by the World, Requirements, Specifications, Program, and Machine model [62] and the Four-Worlds model [63], there are four kinds of contexts:

- the *requirements document*, i.e., the RS, that contains the requirements sentence,
- the *application domain* of the CBS specified by the RS, i.e., the CBS's organizational environment and the behaviors of the CBS's external agents,
- the *system domain* of the CBS specified by the RS, i.e., the conceptual models of the CBS and the models' behavior, and
- the *development domain* of the CBS specified by the RS, i.e., the conceptual models of the CBS's development processes and products.

Therefore, a software-engineering ambiguity is of at least one of four kinds, each named after a context. The context that is relevant for identifying an ambiguity is independent of the context that might be needed to disambiguate the ambiguity, e.g., application domain information may be needed to disambiguate a requirements document ambiguity. In fact, application domain information, obtained from the CBS's client or users, is often needed to disambiguate any kind of ambiguity.

A *requirements-document ambiguity* occurs when a requirement statement in a RS allows several interpretations based on what is known about other requirements in the same RS. A single requirement R is almost never self contained, almost always referring explicitly or implicitly to other requirements in the same document. As a result, the reader must know these other related requirements in order to fully understand R. Thus, a requirements-document ambiguity can arise from a referential ambiguity. In the requirement The product shall show all roads predicted to freeze., the definite noun phrase roads can refer to more than one set of roads defined earlier in the containing RS.

An *application-domain ambiguity* occurs when a requirement allows several interpretations based on what is known about the application domain. Such an ambiguity can be spotted by only a reader that has sufficient domain knowledge. Parnas, Asmis, and Madey [64] give an example of this kind of ambiguity in the requirement sentence: Shut off the pumps if the water level remains above 100 meters for more than 4 seconds. The readers were not told that the water level varies continuously. As consequence of the continuously varying water level, the sentence has at least four interpretations, based on how the current water level is determined; the water level that is compared to 100 meters for the past 4 seconds can be the (1) *mean*, (2) *median*, (3) *root mean square*, or (4) *minimum* water level. The software engineers implementing this requirement assumed the "minimum water level" interpretation when the "root mean square water level" interpretation is required to deal with the sizable, rapidly changing waves in the tank. Interestingly, the interpretation assumed in other engineering areas is the correct one.

A *system-domain ambiguity* occurs when a requirement allows several interpretations based on what is known about the system domain. The requirement, If the timer expires before receipt of a disconnect indication, the SPM requests transport disconnect with a disconnect request. The timer is cancelled on receipt of a disconnect indication. is ambiguous because it cannot be determined strictly from the requirement's sentential structure if the second sentence is part of the response to the condition following the If in the first sentence. A bit of domain knowledge tells the reader that cancellation of an expired timer makes no sense, and therefore, the second sentence is *not* part of the response to the condition.

A *development-domain ambiguity* occurs when a requirement allows several interpretations based on what is know about the development domain. The sentence The doors of the lift never open at a floor unless the lift is stationary at that floor. is ambiguous because it cannot be determined from the sentence alone whether the sentence is a requirement to be implemented in the CBS or the sentence is a statement of what the CBS can assume to be true of the lift hardware. In other words, it is not known whether the sentence is optative or indicative [53]. Someone understanding the development context and knowing a bit about the specific lift hardware chosen for the building can disambiguate the sentence. If this sentence were to occur in a RS meeting U.S. Government SRS standards, then the sentence would be regarded as indicative since an optative sentence must have the verb "shall" to indicate that the sentence gives a requirement and is not making a statement about the environment.

Language-Error Ambiguity Berry, Kamsties, and Krieger have identified another category of pragmatic ambiguity, *language error* [12, 13]. As is the case with the other categories of ambiguity, language error may not be mutually exclusive of other categories. A language error ambiguity occurs when a grammatical, punctuation, word choice, or other mistake in using the language of discourse leads to text that is interpreted by a receiver as having a meaning other than that intended by the sender.

For example, Every light has their switch. has a grammatical error that is commonly committed by present-day, even native, English speakers. The error is that of considering every X, which is singular, as plural although it precedes a correct singular verb, as in Everybody brings their lunch. In the case of Every light has their switch. the reader does not know if the intended meaning is "Every light has its switch.", that is, "Each light has its switch.", or is "All lights have their switch.", which could mean either of: "All lights share their switch." or "Each light has its own switch." Basically, because of the error, the reader does not know how many switches there are per light.

Many times, a language error ambiguity is at the same time another kind of ambiguity, especially an *extension versus intention* ambiguity. That is, the sender does not know an error has been committed, and the receiver may or may not know that an error has been committed. If the receiver does not know, she may or may not understand it as intended. If she does know, she may or may not be able to make a good guess as to what is intended, but in the end, she may be left wondering.

The reason this new category is needed is that sometimes there is a language error, but no extension versus intention ambiguity. Sometimes, there is a linguistic mistake only if the intention is one way but not if it is another way. For example, in Everybody brings their lunch., everyone knows that the intended meaning is "Everybody brings his lunch." even though their, being plural, is incorrectly used with the singular Everybody; here we have a language error without an extension versus intention ambiguity. However, if their refers to a plural noun in a previous sentence, then there is no language error and no real ambiguity. Nevertheless, the reader may have forgotten the plural noun, and she may interpret the their as referring, although with a grammar error, to Everybody.

In I only smoke Winstons., if the intention is to say, “I smoke only Winstons.” there is the language error of a misplaced only. However, if the intention is to make it clear, in an admittedly strange conversation about eating Winston cigarettes, that one only smokes and does not eat Winstons, then there is no language error. However, someone not privy to the whole conversation, and hearing only I only smoke Winstons. may understand “I smoke only Winstons.”, which would be contrary to the intention, even though the intention is in fact what is said by the sentence, according to the rules about placement of only.