A Fragmentation Technique for Parsing Complex Sentences for Machine Translation

by

Jung-Taik Hwang

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the Degrees of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1997

Author _____
Department of Electrical Engineering and Computer Science
May 23, 1997

Certified by _____
Clifford J. Weinstein
Group Leader, MIT Lincoln Laboratory
Thesis Supervisor

Certified by _____
Young-Suk Lee
Staff, MIT Lincoln Laboratory
Thesis Supervisor

Accepted by _____
Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

A Fragmentation Technique for Parsing Complex Sentences for Machine Translation

by

Jung-Taik Hwang

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 1997, in partial fulfillment of the
requirements of the degree of
Masters of Engineering

Abstract

The Speech Systems Technology Group at MIT Lincoln Laboratory has been developing an automatic speech and text translation system called CCLINC, for English to Korean translation. For the system to be useful in real world applications, it has to be robust enough to translate any new input sentence. A major challenge in developing a robust translation system lies in the parsing of a complex sentence containing multiple clauses and coordinates. A complex sentence induces a higher degree of ambiguity than a simple sentence, and poses a problem for both processing and grammar acquisition of a complex sentence input. A method of overcoming such a problem by means of sentence fragmentation, including a fragmentation algorithm and experimental results, is presented in this thesis. The algorithm utilizes and builds on the output of the Apple Pie Parser developed by Satoshi Sekine at New York University. In addition, a possible future application of a fragmentation technique to locate noun phrases for purposed of information retrieval is discussed.

Thesis Supervisor: Clifford J. Weinstein
Title: Group Leader, MIT Lincoln Laboratory

Thesis Supervisor: Young-Suk Lee
Title: Staff, MIT Lincoln Laboratory

Acknowledgments

Many individuals have helped me write this thesis both directly and indirectly. I would like to start off my thanking my academic advisor Albert Meyer. He provided me with invaluable advice in both academics and outside of academics. I would also like to thank my thesis supervisor and Group Leader at MIT Lincoln Laboratory, Clifford Weinstein. Without Cliff, I would have never had an opportunity to work on this project. I want to also thank Young-Suk Lee, my other thesis supervisor, for everything she has done for me. Without her, this thesis would be no where close to being done. She took care of everything from making sure I had interesting research work, to coming up with a timetable for my thesis work, to providing me invaluable feedback on my work. Thank you Young-Suk.

Next, I would like to thank all the members of Group 49 at Lincoln. I am glad to say that I had a wonderful time working here. I would like to especially thank Linda Kukolich for keeping her door open for the many conversations about anything and everything. I would like to also say thank you to the members of the Translation Project (Clifford Weinstein, Young-Suk Lee, Dinesh Tummala, Stephanie Seneff, and Linda Kukolich) on whose work my thesis work is based on.

I would like to thank my Theta Xi fraternity brothers and my other special friends (you know who you are) who have helped me survive and get the most out of the MIT experience. What I have learned in school is trivial to the things I have learned from living, breathing, and drinking beer with you. Thank you my friends.

Finally, I would like to thank my parents. Without their financial support, moral support, and love, none of this would have been possible.

Contents

List of Figures

List of Tables

Chapter 1

1 Introduction

The Speech Systems Technology Group at MIT Lincoln Laboratory has been developing an automatic text and speech translation system for English/Korean translation. The system I have been working with takes an English sentence as the input and generates a Korean sentence as the output. The existing system handled shorter simple sentences very well. However, with longer complex sentences containing multiple clauses and coordinates, the system often could not parse the sentence which leads to no translation output at all. To tackle this problem, we looked at techniques of fragmenting the sentence in such as way that the pieces can be translated and then combined at a later stage. This technique enables the translation of relatively long sentences, which poses a challenge to many existing machine translation systems. This chapter gives a quick overview of the current machine translation system, presents the thesis goals, and explains two research domains of interest.

1.1 Introduction To The Lincoln MT System

The goal of the Lincoln project is to automate translation of text and speech for

enhanced C4I[1] among allied military forces, as explained in Reference [5]. The main focus is the translation of English to Korean, however the system has to be flexible enough to extend to other languages with minimal duplication of work. The problem is that human translators in the military spend an enormous amount of time translating documents, messages, and other material. The goal of this research is to increase the effectiveness of human translators by providing them a tool which replaces the manual labor required in translation.

In the process of achieving these goals, current research advances in language understanding, language generation, and speech recognition were adapted. These research areas were integrated into an useful real world application. The main focus of the application is translating messages in the military domain from English to Korean. The results is a system that utilizes robust translation techniques to handle new words and complex sentences. In addition to these techniques, the application also features a graphical user interface for the translator's aid, which is the final application that the translators will use.

The goal was to develop a high performance translation system for translating military task domain. The system was designed to handle highly telegraphic military messages as well as long and complex grammatical sentence. This translation system is called CCLINC.

---

[1]Command, Control, Communications, Computing, and Intelligence

1.2 CCLINC System

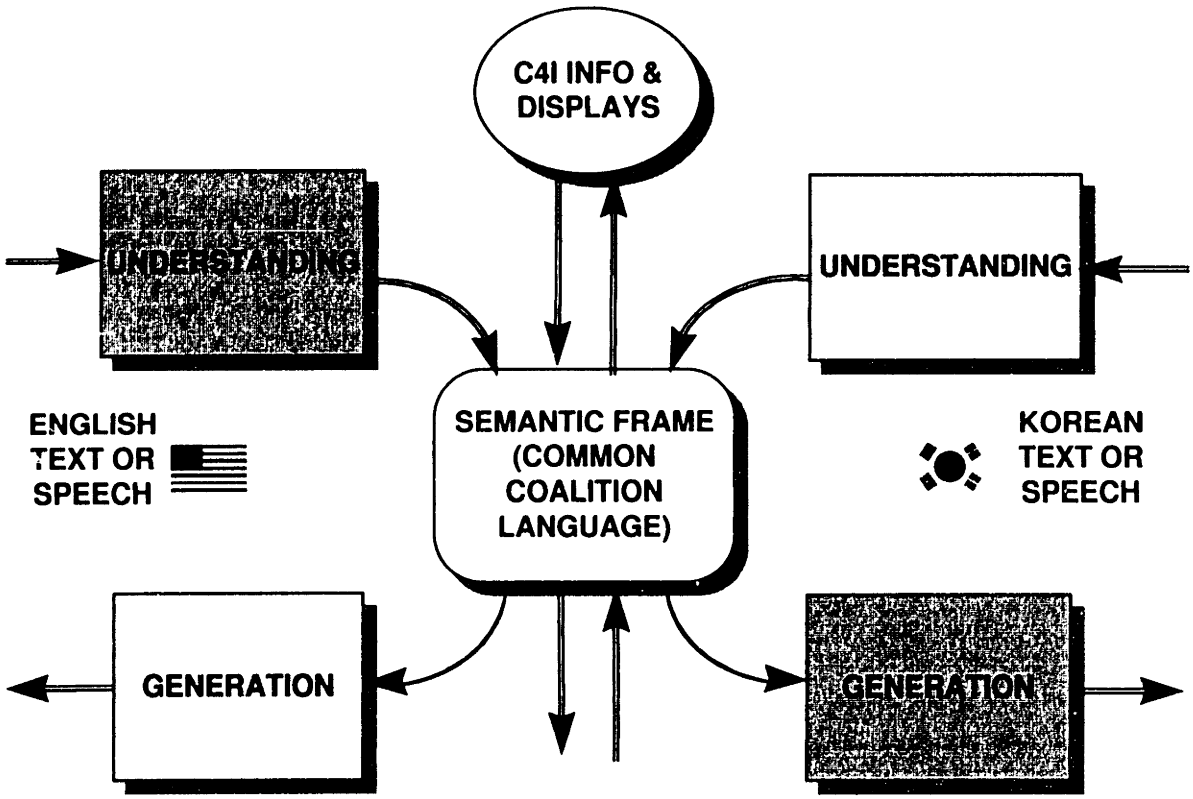

Figure 1-1 CCLINC System Structure

The CCLINC system uses a semantic frame or common coalition language as the common basis for translation. A given text in a source language is translated into the common coalition language through the use of a language understanding module. Then, the text in the common coalition language is translated into the target language through the use of the language generation module. A graphical representation of the system structure is given in Figure 1-1 from Weinstein, Lee, Seneff, and Tummala [13]. This design allows the system to be extended to any other languages by developing the appropriate language understanding and language generation modules.

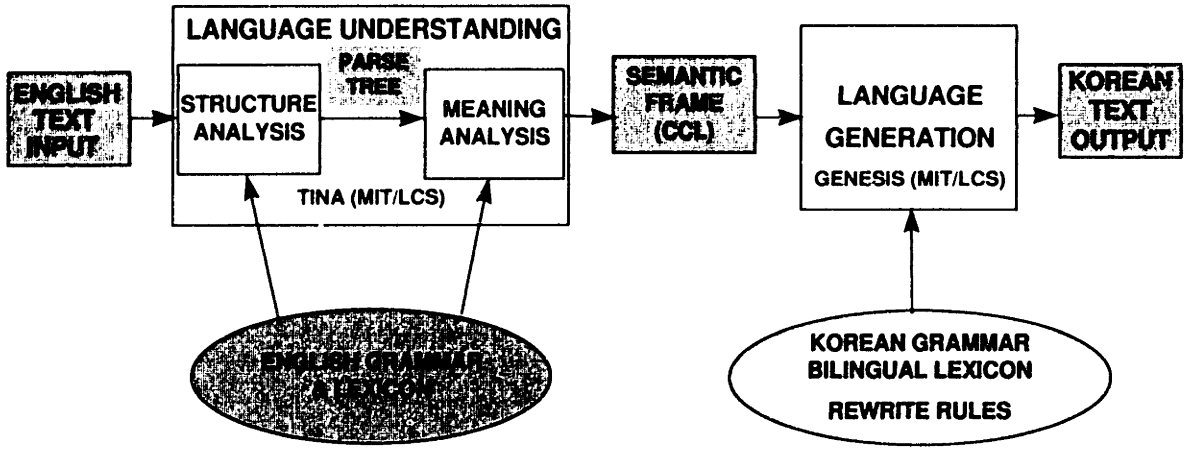

Figure 1-2 Translation System Process Flow

To understand how the system truly works, it is necessary to follow a sentence through the entire translation path. Figure 1-2 from Weinstein, Lee, Seneff, and Tummala [13], shows the original system before the thesis work has been integrated.

To clearly see the process, an example sentence can be followed through the translation pipeline. The following is a typical MUC-II[2] Naval Operations Report message.

INPUT SENTENCE:

0819 Z USS Sterett taken under fire by a kirov with SSN-12's

TINA, the language understanding module does a structural analysis of the input sentence using its English grammar and lexicon. The output of this analysis is a corresponding parse tree. The parse tree that is generated from the example sentence is shown below in Figure 1-3, also from Reference [13].

[2]MUC-II stands for 2nd Message Understanding Conference. MUC-II messages were collected and prepared by NraD to support DARPA sponsored research message understanding. We utilized these messages for the DARPA sponsored machine translation project.

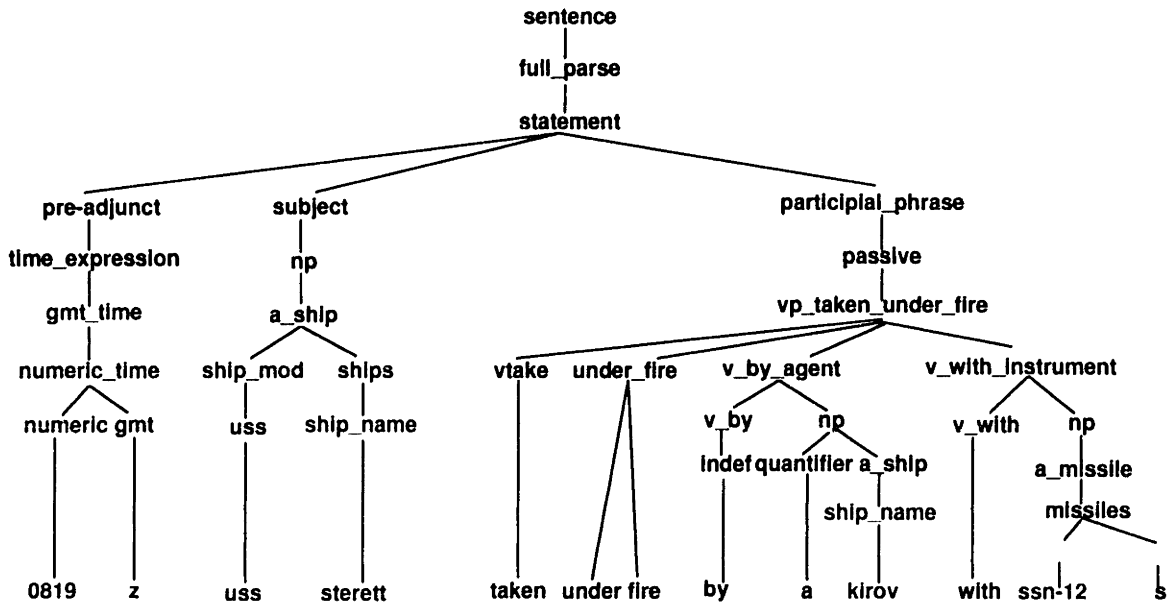

Figure 1-3 Parse Tree Generated by TINA

From the parse tree, TINA generates a common coalition language or semantic frame, though the use of a mapping algorithm. The corresponding semantic frame is shown in Figure 1-4 from Reference [13].

```
{ c statement
    :time_expression {p numeric_time
                        :topic {q gmt
                                  :name "z" }
                        :pred {p numeric
                                  :topic 819 } }
    :topic {q ship_name
              :name "sterett"
              :pred {p ship_mod
                        :topic "uss" } }
    :subject 1
    :pred {p taken_under_fire
              :mode "pp"
              :pred {p v_by
                        :topic {q ship_name
                                  :name "kirov" } }
              :pred {p v_with
                        :topic {q submarine_name
                                  :name "ssn-12s" } } } }
```

Figure 1-4 Semantic Frame

The semantic frame captures the core meaning of the original input sentence. The relationship among all the elements involved is completely captured by the semantic frame. It does this by assigning a specific descriptive label to each word in the parse tree. For example ssn-12 is correctly represented as a submarine_name in the statement instead of just as a noun. To generate the Korean text output, the language generation module, Genesis, takes the semantic frame as the input and produces the Korean text by using generation rules that are unique to the target language.

1.3 Thesis Goals

In this thesis, I worked on methods to improve our existing translation system through the use of a sentence fragmentation technique. The problem was that the existing system did not deal very well with complex sentences (sentences with more than one clause) as well as other long sentences. This is a common problem in many existing machine translation systems because complex sentences introduce a higher level of ambiguity in the parsing stage. However, since TINA, the language understanding module, performs very well with simple military domain sentences and the entire system is stable, we wanted to add a module to act as a pre-processor to TINA rather than making the sentence fragmentation an internal component of it.

The overall goal of the research was to integrate an accurate Sentence Fragmenter into the existing system in order to improve performance on complex sentences. The solution involved pre-processing the sentence before it is passed to the language understanding system and breaking up the sentence into smaller fragments. The smaller fragments are then

fed through the language understanding system and the results are recomposed together at the semantic frame level, while preserving the meaning of the original sentence.

There were three goals that I wanted to accomplish in this thesis. The first goal was to understand the inner-working of the Apple Pie Parser so we can use it as a starting point in the research, which is addressed in chapter 2 of this thesis. The second goal was to design and implement a Sentence Fragmenter algorithm, which is presented in chapter 3. The algorithm has gone through many iterative steps over the period of the thesis in order to improve performance. The third goal was to integrate the new Sentence Fragmenter into the overall system, which is presented in chapter 4. Finally, in chapter 5, I suggest ways of improving the performance of the Fragmenter and some potential applications of the sentence fragmentation technique in other research areas.

I am very happy to have been able to successfully achieve all three of the goals mentioned above. However, in a pioneering research area as this one, further research can always be used to improve the existing work. Specifically, the work accomplished in this thesis shows the feasibility of using the technique of sentence fragmentation for coping with the problem of complex sentence. However, the sentence fragmentation algorithm is far from perfect. In this thesis, in addition to the current work, I also propose several possible ways that the system can be improved in the future.

1.4 Research Domain

As stated earlier, the goal of the project is to develop a machine translation system for a military task domain. Specifically, the domains we are interested in translating are

military domains. The two main message categories that we are interested in translating are MUC-II data and Command and Control Warfare (C2W) data.

MUC-II data is messages used in Naval Operations Reports. They are often highly telegraphic, containing a lot of domain specific elliptical expressions. MUC-II data was the original target of the machine translation project, which over time has been extended to handle other types of data. Below are a few examples from MUC-II data and a paraphrase of the intended message. The telegraphic nature of MUC-II data is very clear from these two examples.

```
SENTENCE: 3a-6e's launched agm 84's foll by 2 a-6e's launch of skipper iii

PARAPHRASE: Three A-6e aircrafts launched AGM 84 missiles and was followed
by two A-6e aircrafts launching skipper III missiles.

SENTENCE: friendly cap a/c splashed hostile tu-16 proceeding inbound to
enterprise at 35nm

PARAPHRASE: Friendly combat air patrol aircraft splashed a hostile TU-16
aircraft which was proceeding inbound toward USS Enterprise when it was 35
nm away
```

Another domain is Command and Control Warfare domain (C2W). The sentences forming this domain were extracted from the Command and Control Warfare handbook, which is a bilingual book written in both Korean and English. This is a great source of a large amount of bilingual text which is otherwise very scarce. C2W data differs from MUC-II data in the sense that since it is extracted from a handbook, it uses grammatical correct English sentences. Below are a couple of sentence from the C2W data.

```
military deception can be used to enhance the impact obtained from
practicing these tenets, but military deception also provides
opportunities to apply these tenets in ways that are not otherwise
available

electronic deception, as another component of ea, conveys false or
misleading information to enemy decisionmakers to influence their
perception of an operational situation incorrectly and respond, or fail to
respond, in a way advantageous to friendly operations
```

Initially, the sentence fragmentation work was started using the MUC-II data. However, due to the telegraphic nature of the MUC-II, the Apple Pie Parser's performance was poor because it was trained using normal grammatically correct sentences. Therefore, basing the sentence fragmentation algorithm on MUC-II data didn't seem like the best choice.

Upon coming across C2W data, we decided to use that data as a basis of the thesis work because of several properties of these sentences. First, they were grammatically correct which translated into a better performance of the Apple Pie Parser. Second, the sentences were very long, containing many clauses and coordinates, which is what we needed to develop and test the framentation algorithm. The average length of the 286 test sentence from the C2W domain is 15 words[3], however many of the sentences used in the research exceed 25 words.

1.5 Summary

This thesis presents a sentence fragmentation technique to overcome the problem of parsing complex sentences. The current understanding system, TINA, is a system designed to handle domain specific grammar and achieves a high performance for simple sentences in the MUC-II domain. However, when presented with longer sentences such as sentence from the C2W domain, TINA's grammar is unable to handle them.

The overall goal of the thesis is to solve this problem through the use of sentence fragmentation. In achieving this overall goal, several sub goals were established. These are

[3]See Reference [6].

to learn the inner workings of the Apple Pie Parser, build a Sentence Fragmenter based on that knowledge, and integrate the research into the existing system.

Chapter 2

2 Apple Pie Parser

The Sentence Fragmenter is built on the output of the Apple Pie Parser developed by Satoshi Sekine at NYU. Therefore, it is necessary to understand the inner workings of the Apple Pie Parser in order to efficiently implement the Sentence Fragmenter.

The following chapter describes the inner workings of the Apple Pie Parser. This includes an overview of the parser in Section 2.1, the bottom-up chart parsing technique in Section 2.2, the probability calculations in Section 2.3 and Section 2.4, and the technique of fitted parsing in Section 2.5. In addition, experimental results and the drawbacks of the system are presented in Section 2.6.

2.1 Overview of the Apple Pie Parser

The Apple Pie Parser Version 5.8 was developed by Satoshi Sekine at New York University. It is a bottom-up probabilistic chart parser which uses a best-first search algorithm to assign scores to the individual parse trees. The grammar of the parser is a semi-

context sensitive grammar with two non-terminals[1].

The Apple Pie Parser's grammar was extracted from Penn Tree Bank, which is a syntactically tagged corpus at the University of Pennsylvania. The Penn Tree Bank corpus contains over 47000 sentences gathered from the Wall Street Journal. The grammar was acquired fully automatically from the Penn Tree Bank. The advantage of this method was that it was possible to generate a very broad coverage grammar without having to invest a large amount of human labor.

The output[2] of the Apple Pie Parser is a syntactically bracketed sentence as given in the example below.

Input:

```
the damaged enemy bogey left the friendly area
```

Output:

```
(S (NPL the damaged enemy bogey) (VP left (NPL the friendly area)))
```

The parsed sentence has syntactic brackets as well as syntactic categories[3] for the corresponding part of the sentence.

2.2 Bottom-Up Chart Parsing

The Apple Pie Parser uses a bottom-up chart parsing method. A bottom-up strategy involves starting with the words in the sentence and using rewrite rules to reduce the

---

[1]The two non-terminals are **S** and **NP**.

[2]The Apple Pie Parser is capable of generating four different types of outputs. Refer to the Manual of the Apple Pie Parser (Reference [9]) for more details.

[3]A list of labels can be found in the Appendix.

sequence of symbols until it terminates with a single S, the top level symbol for a sentence. Contrarily, a top-down strategy involves starting with a single S and searching for different ways to rewrite the symbol until the input sentence is generated. Figure 2.1 illustrates both the bottom-up and top-down strategies.

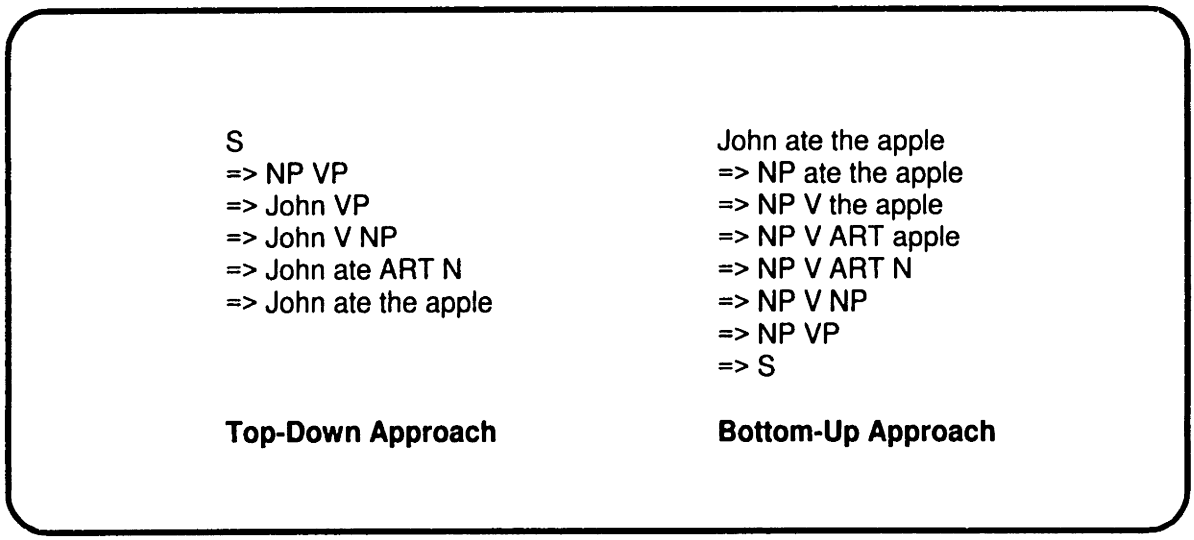

Figure 2-1 Top-Down and Bottom-Up Parsing Methods

The bottom-up chart parsing technique uses the technique of a bottom-up approach with the addition of one performance enhancing optimization to the algorithm. The technique uses the concept of a "chart" structure in order to store partial results so that it can recall them for later use instead of recomputing the partial results each time they are used. A step by step derivation is given later in this chapter, when the probability calculations are introduced into the bottom-up chart parser.

A bottom-up chart parser uses the following strategy[4]. The parser maintains an agenda, a chart, and a list of active nodes. The agenda contains a list of constituents that have yet to be added to the chart, which contains partially constructed constituents, and the list of

[4]For a detailed description of the bottom-up chart parsing method, see reference [1]

active nodes contain grammar rules and the current positions within the rules that are currently being considered. The algorithm reads one word at a time, keeps track of the possible rules that the word introduces and attempts to combine constituents to form new constituents. The sentence is completely and correctly parsed when the agenda contains only the top-level symbol and there are no more word left in the sentence. The storing of the partial result is achieved through the use of active nodes by allowing the algorithm to examine each word only once.

Because the rules were acquired automatically from a tree bank, there are over 32000 grammar rules in the system, approximately 9000 of which are rules for S, the top level symbol in a sentence. The very large number of grammar rules, combined with the fact that the Apple Pie Parser grammar only has two non-terminals, eliminates the possibility of using a simple top-down or bottom-up approach because such methods would generate an enormous number of active nodes in the parsing process.

Even the bottom-up chart parsing method generates too many active nodes for practical purposes. This problem is solved by the author of the Apple Pie Parser by integrating some strategies which reduce the number of active nodes. One strategy for reducing the number of active nodes is by factoring the grammar rules with common prefixes.

```
RULE 1: NP -> ART N
RULE 2: NP -> ART ADJ N
RULE 3: NP -> ART ADJ ADJ N
RULE 4: NP -> ART N PP

PHRASE: a large dog
```

Figure 2-2 Example of Factoring Grammar Rules

Suppose we have four rules and the sequence of words to be parsed given in Figure 2-2. Upon reading the article *a*, the Apple Pie Parser would only generate one active node instead of the normal four. The active node would contain a dot after ART and a list of four pointers pointing to the individual rules. Hence, we have factored the four rules into one which uses less memory. Instead of maintaining all the rules that have common prefixes separately in the list of active nodes, the Apple Pie Parser simply maintains a single pointer to that node in the grammar automaton.

Furthermore, if there are still too many active nodes, the Apple Pie Parser resorts to a secondary backup grammar which contains fewer grammar rules. This is the second strategy used to lessen the memory requirements.

2.3 Probability in Lexicon and Grammar

As mentioned before, the Apple Pie Parser is a probabilistic parser. It uses probability in two distinct ways. First, it uses probability to choose which grammar rule to use in case it runs into an ambiguity. Second, it uses probability in the lexicon to determine the likelyhood of a word being a certain part of speech. For example, the word *can* can be used as a noun, a main verb, or an auxiliary verb. The probability measures are then used to calculate the weight or the "validity" which is used to resolve ambiguities.

Using probability in parsing techniques is similar to the previous example because we are interested in parsing sentences that we have not encountered before. The goal is to use the sentences of the training set to predict the interpretation of the new test sentence.

Probability is first used to calculate the probability in the lexicon. For example, when

the Apple Pie Parser comes across the word *can*, it must decide between whether it is used as a verb or a noun. Since both the grammar and lexicon was derived from the Penn Treebank project, the probability assigned to the particular word depends on the way it was used in the Treebank. As it turns out, *can* was used as a verb 1079 times and as a noun only 6 times. Therefore, the parser guesses that for the given word in our sentences, there is a 1079/1085 = 99.45% probability that it will be used as a verb. Since the word has been seen 1085 times in the corpus, we can be fairly confident in the accuracy of our probability calculation.

In contrast, take the word *bend* for example. In the training data, it was used as a verb five times and as a noun once. As we did before, we can determine that the Apple Pie Parser will treat *bend* as a verb with 5/6 = 83.33% probability. However, we can reason that this probability measure has a greater margin for error because it is based on a fewer number of samples, six, compared to the previous example which was seen 1085 times in the corpus. In general, since the probability for a given word is only used as a small portion of the whole probability calculation, the performance of the lexical probability calculator is pretty good. The main reason behind the good performance is that most words have only one interpretation in the dictionary.

The performance of the part of speech tagger can be significantly improved by using a more complex strategy. For example, we can use the words around the word we are examining to derive a more accurate probability. However, the Apple Pie Parser uses the simple strategy of assigning tags for complexity and speed reasons. Therefore, if the first assigned tag generates a lower score after the grammar rules are applied than a less common interpretation, the parser will actually use the less common interpretation. The formula of

how the probabilities for the lexicon and the grammar are related is examined in the next section. We will also examine errors caused by mistagging, a situation where the incorrect tag was selected by the parser, later in the paper.

Rule	Count for LHS	Count of Rule	Probability
1. S->NP V	1000	800	0.8
2. S->NP V NP	1000	200	0.2
3. NP->NP PP	3000	750	0.25
4. NP->N N	3000	300	0.1
5. NP->N	3000	1000	0.333
6. NP->ART N	3000	950	0.317

Figure 2-3 Simple Grammar Rules

The second way probability is used in the Apple Pie Parser is in the grammar. The way that the Apple Pie Parser uses probability in the grammar is that it uses the number of constructions seen for each rule in the training data. Then by using the probabilities, it then finds the parse tree that could have generated the sentence. Together with the probabilities acquired from the lexical analysis, the parser uses the grammar probabilities to return the most-likely parse tree.

2.4 Probability Calculation

The Apple Pie Parser uses the following strategies for assigning probabilities. In assigning part of speech tags, it uses a simple probability function.

$$P_{tag}(t \mid w) = \frac{\textit{Frequency of word w with tag t}}{\textit{Frequency of word w}}$$

To calculate the score of a parse tree S_{tree}, the following formula is used. The probability for a given grammar rule X -> Y is based on the frequency with which X dominates Y in the training corpus and the frequency of X. For example, in Figure 2-3, the probability of the rule [NP -> N N] is 0.1 because NP appears 3000 times in the training corpus and the right hand side construction appears 300 times.

$$P_{rule}(X -> Y) = \frac{\textit{Frequency with which X is expanded as Y}}{\textit{Frequency of X}}$$

The total score for a given parse tree is the product of each rule used to build the tree together with the square of the probability of the tag for each word in the sentence. The square factor acts to put more emphasis on the tag probability over the rule probability which produces better results than without the square factor.

$$S_{tree}(T) = \prod P_{rule}(R) \prod (P_{tag}(t \mid w))^2$$

The best parse tree is the one with the highest score among the trees possibly derived from the input. The probability calculation methods of the Apple Pie Parser can be best seen through the computation of an example.

The bottom-up probabilistic chart parsing can best be shown with a simple example. For more examples, see Appendix C. Consider the simple sample sentence, *the dog chased the cat*. This is interpreted as a list of tokens seperated by node markers, 0 START 1 the 2 dog 3 chased 4 the 5 cat 6 END 7. After the start symbol is read, the parser reads the word *the*. The word is looked up in the dictionary and the tag DT (for a determiner) is assigned since it is the only entry. The second step is to create active nodes from node 1 to node 2. This is done in the Apple Pie Parser by searching for the rule which has DT on the right hand

```
1.     [  0,  1] :    0(  0)  -{   -1}-> =S=
2.     [  1,  2] :   DT(  0)  -{   -1}-> the
3.     [  1,  2] :  NPL( 49)  -{37495}-> DT[1 2]
4.     [  1,  2] :   SS(103)  -{25363}-> NPL[1 2]
5.     [  1,  2] :   NP( 90)  -{17837}-> NPL[1 2]
6.     [  1,  2] :    S(135)  -{ 3259}-> NPL[1 2]
7.     [  1,  3] :  NPL( 20)  -{38966}-> DT[1 2] NNX[2 3]
8.     [  1,  3] :   NP( 61)  -{17837}-> NPL[1 3]
9.     [  1,  3] :   SS( 74)  -{25363}-> NPL[1 3]
10.    [  1,  3] :    S(106)  -{ 3259}-> NPL[1 3]
11.    [  1,  4] :  NPL( 96)  -{39163}-> DT[1 2] NNX[2 3] VBX[3 4]
12.    [  1,  4] :   SS( 58)  -{28334}-> NPL[1 3] VBX[3 4]
13.    [  1,  4] :   NP( 97)  -{20757}-> NPL[1 3] VBN[3 4]
14.    [  1,  4] :    S( 88)  -{ 5358}-> NPL[1 3] VBX[3 4]
15.    [  1,  6] :   SS( 79)  -{28580}-> NPL[1 3] VBX[3 4] NPL[4 6]
16.    [  1,  6] :   NP(114)  -{21318}-> NPL[1 3] VBX[3 4] NPL[4 6]
17.    [  1,  6] :    S( 91)  -{ 5748}-> NPL[1 3] VBX[3 4] NPL[4 6]
18.    [  2,  3] :  VBX( 25)  -{   -1}-> dog
19.    [  2,  3] :   VB( 25)  -{   -1}-> dog
20.    [  2,  3] :  NNX(  1)  -{   -1}-> dog
21.    [  2,  3] :  NPL( 22)  -{41257}-> NNX[2 3]
22.    [  2,  3] :   SS( 76)  -{25363}-> NPL[2 3]
23.    [  2,  3] :   NP( 63)  -{17837}-> NPL[2 3]
24.    [  2,  3] :    S(108)  -{ 3259}-> NPL[2 3]
25.    [  2,  4] :  NPL(109)  -{41467}-> NNX[2 3] VBX[3 4]
26.    [  2,  4] :   SS( 60)  -{28334}-> NPL[2 3] VBX[3 4]
27.    [  2,  4] :   NP( 99)  -{20757}-> NPL[2 3] VBN[3 4]
28.    [  2,  4] :    S( 90)  -{ 5358}-> NPL[2 3] VBX[3 4]
29.    [  2,  6] :   SS( 81)  -{28580}-> NPL[2 3] VBX[3 4] NPL[4 6]
30.    [  2,  6] :   NP(116)  -{21318}-> NPL[2 3] VBX[3 4] NPL[4 6]
31.    [  2,  6] :    S( 93)  -{ 5748}-> NPL[2 3] VBX[3 4] NPL[4 6]
32.    [  3,  4] :  VBN( 10)  -{   -1}-> chased
33.    [  3,  4] :  VBX(  4)  -{   -1}-> chased
34.    [  3,  4] :  NPL( 96)  -{41473}-> VBX[3 4]
35.    [  3,  4] :   SS( 73)  -{35156}-> VBN[3 4]
36.    [  3,  4] :    S(100)  -{15968}-> VBX[3 4]
37.    [  3,  4] :   NP(164)  -{17824}-> SS[3 4]
38.    [  3,  6] :   SS( 67)  -{35526}-> VBX[3 4] NPL[4 6]
39     [  3,  6] :   NP(131)  -{22827}-> VBX[3 4] NPL[4 6]
40.    [  3,  6] :    S( 92)  -{15975}-> VBX[3 4] NPL[4 6]
41.    [  4,  5] :   DT(  0)  -{   -1}-> the
42.    [  4,  5] :  NPL( 49)  -{37495}-> DT[4 5]
43.    [  4,  5] :   SS(103)  -{25363}-> NPL[4 5]
44.    [  4,  5] :   NP( 90)  -{17837}-> NPL[4 5]
45.    [  4,  5] :    S(135)  -{ 3259}-> NPL[4 5]
46.    [  4,  6] :  NPL( 19)  -{38966}-> DT[4 5] NNX[5 6]
47.    [  4,  6] :   NP( 60)  -{17837}-> NPL[4 6]
48.    [  4,  6] :   SS( 73)  -{25363}-> NPL[4 6]
49.    [  4,  6] :    S(105)  -{ 3259}-> NPL[4 6]
50.    [  5,  6] :  NNX(  0)  -{   -1}-> cat
51.    [  5,  6] :  NPL( 21)  -{41257}-> NNX[5 6]
52.    [  5,  6] :   SS( 75)  -{25363}-> NPL[5 6]
53.    [  5,  6] :   NP( 62)  -{17837}-> NPL[5 6]
54.    [  5,  6] :    S(107)  -{ 3259}-> NPL[5 6]
55.    [  6,  7] :    0(  0)  -{   -1}-> =E=
```

Figure 2-4 Chart Parsing of the Sentence *the dog chased the cat*

side. This rule actually is a factored form of all the rules that have DT as the first element of the right hand side. By just adding one active node for all the rules that begin with DT, the

Apple Pie Parser dramatically reduced the number of active nodes that it needs to keep track of. In the traditional chart parsing technique without this factoring processes, there would be many more active nodes that would be generated by the first token. Figure 2-4 line 3 represents that there exist rules for NPL which start with the symbol DT (this rule has a score of 49). Then, the parser looks up all the rules that can be composed from the constituents in the chart, which at this step is only the symbol NPL. The parser generates three active nodes that has NPL as the righthand side. These rules are shown in lines 4,5,6 in Figure 2-4. After these three active nodes are added, the parser is done with the first step. Since chart parsing does not require back tracking, it isn't necessary to revisit the first *the* later in the parsing process.

To start the next step, the word *dog* is read and looked up in the dictionary. The word *dog* has been seen 11 times as a NN, once as a VB and once as a VBP in the training data. Using the NN defintion, the active node shown in line 3 is extended from [1-2] to [1-3] to generate a new active node shown in line 7. From this new active node, which has the label NPL on the left hand side, the new nodes shown in lines 8, 9, and 10, which have NPL as the first element on the right hand side are created. In addition, new active nodes covering [2-3] are created in similar fashion to how active nodes for [1-2] were created. These are shown in lines 21, 22, 23, and 24. From the chart, it can bee seen that there were no rules that has VB or VBP as the first element on the right hand side.

Next, the word *chased* is looked up in the dictionary, which shows that it has been seen as a VBN once and a VBX twice. As with before, new actives nodes 34, 35, 36, 37 spanning [3-4] are created. Then, the active node shown in line 7 is extended from [1-3] to [1-4] to create active node shown in line 11, using the VBX definition. Similarly, using the

VBX definition, the actives nodes shown in lines 8, 9, and 10 are extended from [1-3] to [1-4], to create the new actives nodes in lines 12, 13, and 14. Using the same technique, the actives nodes spanning [2-3] are extended to [2-4], shown in lines 25 26, 27, and 28.

The fourth token, the second *the* is looked up in the dictionary to be a DT. As before, new actives nodes [4-5] are generated, shown in lines 42, 43, 44, and 45. Then the parser tries to extend the existing active nodes by using the new token. None of the active nodes spanning [1-4] or [2-4] or [3-4] can be extending by the token DT, therefore no new actives nodes are generated in the extending stage.

The last token, *cat* has only a NNX interpretation in the dictionary. As usual, this generated some active nodes from [5-6] shown in lines 51, 52, 53, and 54. In the extending stage, only the active nodes spanning [4-5] can be extended. This step is identical to the extension from [1-2] to [1-3]. Four new active nodes shown in lines 46, 47, 48, and 49 are created. However, we can now use the constituent spanning [4-6], namely NPL, to extend other active nodes which terminate at node 4. This creates new active nodes spanning [1-6] (lines 15, 16, and 17) and spanning [2-6] (29, 30, and 31).

Finally, the END symbol is encountered, marking the end of the sentence. At this point, the parser looks at all active nodes that span the whole sentence ([1-6] in the case) that also have S as the left hand side. The rule with the lowest[5] score is chosen at this point, although in this example, there is only one active node which meets the criteria. This is grammar rule number 5748 shown in line 17 of Figure 2-4.

```
[  1,  6] :    S( 91)  -{ 5748}-> NPL[1 3] VBX[3 4] NPL[4 6]
```

[5]The lowest score represents the highest probability due to the log function and negative constant factor used in the calculation, as explained later in the chapter.

Rule 5748 in the grammar file is:

```
1 : 272 483 272
 :score 48
  :struct  "(S <1> (VP <2> <3>))"  ;
```

From this, the parser replaces <1> and <2> by our words and returns the following output.

```
(S (NPL the dog) (VP chased (NPL the cat)))
```

The score calculation for the given node is as follows. The score of 91 given in line 17 was generated by the following step by step derivation. The rule itself has a score of 48. In addition to this, the same rule which coverts *the dog* and *the cat* into NPL (rule number 38966) has a score of 19. In addition, the rule which treats *dog* as a NNX has a score of 1 and the rule which treats *chased* as a VBX has a score of 4. Summing them, 48 + 19 + 19 + 1 + 4, we get a composite score of 91.

The scores are added instead of multiplied because the author takes the logarithm of the probabilities. This natural logarithm function is already pre-applied to the grammar rules. To see how this works in the dictionary, consider how the scores of 10 and 4 are assigned to the interpretations VBN and VBX for the word *chased* respectively. From the dictionary, we can see that *chased* was seen twice as a VBX and once as a VBN in the training set. This sets the probability of "chased being used as VBX" at 0.666 and the probability of "chased being used as a VBN" at 0.333. The scores can be derived from the probabilities using the following formula.

Score for VBN = -10.0 * ln (0.333) = 10.996 which is truncated to 10.

Score for VBX = -10.0 * ln (0.666) = 4.065 which is truncated to 4.

The 10 is the weight that is placed on the part of speech tagging and a factor that is adjustable in the parameter file. The negative sign is used to convert the score into a positive

number. The author uses the log function to transform all probabilities (both lexical and grammar) because logarithms allows the multiplication problem to be transformed to an addition problem. Logs have a unique feature that the product of any two numbers equals the sum of the logs. Therefore, instead of using multiplication functions throughout the process as new words and grammar rules are used, by using this transformation, the author manages to simplify the process. By using the score assignment strategy describes above, the score reflects the probabilities of both the lexical assignments as well as the grammar rules used in the construction.

2.5 Fitted Parsing

One major reason that we chose to use the Apple Pie Parser as a base is that it produce an output for any sentence regardless of the construction of the sentence. Although there are over 30000 rules in the grammar of the Apple Pie Parser, it is still not possible to produce parse tree for all possible sentences. In these cases, the Apple Pie Parser "gives up" and uses a technique called fitted parsing to return the best result it can.

Fitted parsing occurs when the parse tree cannot be constructed due to the fact that no grammar rules cover the whole input sentence. In this case, the Apple Pie Parser returns a partial result or a fitted parse. The basic strategy is to identify parts of the input structure that can be reliably identified. The list of these "fitted" labels that are allowed are listed in the parameter file.

Generally fitted parsing happens with long sentences where specific rules cannot compose the whole sentence. Long sentences tend to generate a larger number of active

nodes and often caused the Apple Pie Parser to run out of memory. When this occurs, as mentioned earlier, the parser resorts to using a smaller list of grammar rules that were seen two or more times in the training data. However, this has the side effect that the grammar is not very complete and a lot of sentence cannot be parsed. However, this is also possible in heavily ungrammatical sentences containing not many nouns[6].

Fitted parsing involves dividing up the sentence into smaller, more manageable pieces. The parser selects sentential coordination boundary markers such as commas that separate possible pieces which individually reduce to an S structure. The markers to be used as sentential boundaries are specified in the parameters file. These include some punctuation marks single quote, brackets (both open and close), comma, period, and colon. It also includes CC (which includes the word *and)* and apostrophe.

In the 200 test sentences, 20 were parsed through fitted parsing. Of those, 6 were correctly parsed because the markers were indeed top level S sepearators. However, in the 14 other cases, the markers were actually only separating higher level structures and were misinterpreted as sentential separators. Considering fitted parsing is a last resort for sentences which do not conform to the grammar rules, the 30 percent accuracy is very good. On the other hand, the other 70 percent of the sentences are completely misparsed and are not salvageable.

The internal workings of a fitted parse is clearer through the use of an example. The detailed chart parsing results can be found in appendix C. Consider the following sentence:

```
military deception can be used to enhance the impact obtained from
practicing these tenets, but military deception also provides
```

[6]If there are many nouns in the sentence, the NP and S terminal grammar would be able to break up the sentence into smaller pieces.

```
opportunities to apply these tenets in ways that are not otherwise
available
```

The Apple Pie Parser tries to parse this sentence using the full grammar. However, due to the long length, the sentence generated more than the allowed 600000 active nodes. The parse then resorts to the much smaller backup grammar, which fails to cover the whole sentence.

At this point, the Apple Pie Parser looks for sentence level coordinators defined in the parameter file. Upon finding *comma*, the Apple Pie Parser treats the fragment before the comma *military deception can be used to enhance the impact obtained from practicing these tenets* as a sentence and parses it. Then it treats the whole segment as an SS structure and ignores any other possible interpretations of the first part. Normal parsing then resumes with the next word which is the word *but*.

2.6 Experimental Results

There are two widely used methods for measuring performance, recall and precision. Recall is defined to be the accurate predictions divided by the correct answer. Therefore, recall is not affected by extra predictions that might be inserted by the parser. Precision is defined as the accurate predictions divided by the total number of predictions. Therefore, precision on the other hand is affected by the extra brackets that are inserted by the parser. A parser that doesn't insert brackets unless it is absolutely certain can have very low recall and very high precision. A parser that inserts brackets very aggressively can have very high recall and very low precision. By using both the precision and recall numbers, we can get a fairly good picture on how close the results were to the correct answers.

The performance of the Apple Pie Parser given by the author is that the parser has a recall of 73.43 percent and a precision of 72.61 percent. The author trained the parser with 96 percent of the Penn Treebank corpus and used the remaining 4 percent to derive this accuracy. Since both the training set and the test set were extracted from the Wall Street Journal, the sentence structures and styles of the training and test sets are very comparable, or more concisely the sentences are from the same domain. Thus, it is reasonable to assume that the performance of the Apple Pie Parser wouldn't be quite as good on sample sentences from different domains.

To calculate the performance of the Apple Pie Parser for purposes of sentence fragmentation, I used a less quantitative and more qualitative measurement. I manually went through each output and classified it as to whether it is a correct parse tree or not. The result is a more useful measure of performance at the cost of some accuracy.

To determine the performance of the Apple Pie Parser on sentences from the military domain, I ran an experiment on 200 sentences extracted from chapters 1,3,8 and 11 of the Command and Control Warfare handbook, described earlier in chapter 1 of this thesis. I chose the 200 sentences randomly, with the only requirement being that they were grammatically correct sentences. The sentences used in this experiment can be found in appendix B.

	Correctly Parsed	Incorrectly Parsed
Number of sentences	110	90
Number of fitted sentences	6	14
Average length (in words)	16.9	21.5

Table 2-1 Experimental Results of the Apple Pie Parser on 200 C2W Sentences

As it can be seen in Figure 2-1, only 110 out of 200 sentences, or 55 percent of the sentences were parsed correctly. However, many of the misparses at the Apple Pie Parser level does not translate into an error at the Sentence Fragmenter level. Also, certain errors that do affect the Sentence Fragmenter can be corrected at the Fragmenter level. Therefore the overall performance of the Sentence Fragmenter can be better than the underlying Apple Pie Parser.

For the purposes of sentence fragmentation, we are more interested in the 90 sentences that did not parse correctly. The 110 correctly parsed sentences can be easily handled by the Fragmenter. Our goal is to salvage as many of the 90 sentences as we possibly can by understanding what caused the error.

Type of error	Frequency
Incorrect rule	32
Two non-terminal side effect	4
Incorrect PP attachment	14
Incorrect fitting	14
Incorrect part-of-speech tagging	26
Total	**90**

Table 2-2 Type and Frequency of Parsing Errors

By examining the misparsed sentences, we can categorize the errors into five different categories. In the cases where more than one type of error in a sentence, we care about the most serious error. The five categories are errors due to the use of the incorrect rule, errors due to incorrect part of speech tagging, errors due to the use of the two-non terminals (S and NP) in the grammar, errors due to incorrect fitting, and finally errors due to incorrect attachment of prepositional phrases. Another type of errors which have been avoided by preprocessing the input sentences are errors due to punctuation. Table 2-2 shows the relative frequency of those errors based on the 90 misparsed sentences. The sentences, sub-divided into their error types, are listed in appendix B.

2.6.1 Misparses Due to Use of Incorrect Rule

The most common errors is due to the Apple Pie Parser choosing the incorrect parse tree due to choosing the incorrect grammar construction. These misparses occur most often because of the incorrect ambiguity resolution. Here is an example of such a sentence.

```
at field army and below the cjg3 is the focal point for c2-attack planning
and execution
```

Below is the incorrect parse tree produced by the Apple Pie Parser.

```
(S (PP at
       (NP (NPL field army)
           and
           (NPL below the cjg3)))
   (VP is)
   (NP (NPL the focal point)
       (PP for
           (NPL c2-attack planning and execution))))
```

The correct parse tree is shown below.

```
(S (PP at
       (NPL field army and below))
   (NPL the cjg3)
   (VP is
```

```
(NP (NPL the focal point)
    (PP for
        (NPL c2-attack planning and execution)))))
```

As it can be seen from this example, the structure of the correct parse tree is very different from the incorrect one given by the Apple Pie Parser. Since the parse output is very different from the correct parse and since there is no easy way of algorithmically identifying such sentences[7], there are no easy ways of fixing this error at the Sentence Fragmenter level.

2.6.2 Misparses Due to Incorrect Part of Speech Tagging

The second most common cause of the misparses is incorrect part of speech tagging. These are the cases where the parser assigned the incorrect part of speech tag because using that tag produced the highest probability for the parse tree according to the formula given for S_{tree} in section 2.4. Here is an example of such a misparse.

```
isolated sources of information alone may not be believed
```

This sentences produced the following misparse.

```
(S (VP isolated
       (SS (NP (NPL sources)
               (PP of
                   (NP (NPL information)
                       (ADVP alone))))
           (VP may
               not
               (VP be
                   (VP believed))))))
```

Where as the correct parse tree should have been the following.

```
(S (NP isolated sources
       (PP of
           (NP (NPL information)
               (ADVP alone))))
   (VP may
       not
```

[7]An algorithm can't identify the error because the output parse is a legal parse with a different "meaning" than the obvious one.

```
(VP be
    (VP believed))))
```

The misparse is caused by the word *isolated* being used as a verb rather than an adjective. If we look up the word *isolated* in the lexicon, we can see that the in the corpus, the word *isolated* was used 11 times as a verb and only once as an adjective. There are two ways that these types of problem can be fixed. First, we can simply adjust the weight of the given word in the parameter file by using the SUP_WORD[8] functionality. However, this could be very time consuming because we would have to manually enter a line for each word. Also, this can cause some side-effects which might cause the word *isolated* to be tagged as an adjective rather than a verb in a different sentence where it is actually used as a verb. A possible second solution would be to examine the structure of the misparses and determine certain patterns which might suggest a incorrect part of speech tag. While this solution might be more complex, it has the advantages that it fixes a set of problems and that it can use surrounding lexical information to determine patterns.

2.6.3 Misparses Due to Two Non-Terminal Grammar

Another category of misparses are caused by the parser's two non-terminal grammar. Because S and NP are the only two non-terminals, any nesting or coordination of any other categories must be specified explicitly in the grammar. This can easily be seen by comparing a couple of examples.

```
pigs, sheep, chicken, dogs, lions, and tigers are animals

(S (NP (NPL pigs)
       -COMMA-
```

[8]The SUP_WORD function allows the user of the Apple Pie Parser to override the definition for any word in the dictionary file.

```
        (NPL sheep)
        -COMMA-
        (NPL chicken)
        -COMMA-
        (NPL dogs)
        -COMMA-
        (NPL lions)
        -COMMA-
        and
        (NPL tigers))
    (VP are
        (NPL animals)))
```

Clearly, we can see from this example that the grammar can correctly handle an indefinite number of noun coordinates. This is because there are grammar rules which define NP recursively as compositions of NP. One such rule is rule number 18336 shown below.

```
68 : 272 406 272
 :score 42
  :struct  "(NP <1> <2> <3>)"  ;
```

This is possible because NP is one of the allowed non-terminals. However, consider another example.

```
the chicken ran, sprinted, walked, smiled, and died
```

```
(S (SS (NPL the chicken)
       (VP ran))
   -COMMA-
   (SS (VP sprinted))
   -COMMA-
   (SS (VP walked))
   -COMMA-
   (SS (VP smiled))
   -COMMA-
   and
   (SS (VP died)))
```

In this example, the parser didn't produce the correct parse because the grammar cannot capture the coordinated verbs. This is because verbs are terminals in the Apple Pie Parser and therefore there cannot exist a rule where verbs are recursively defined. Instead, the only way the previous example could have been parsed correctly is if the Apple Pie Parser saw five coordinated verbs in the training data. This would have generated a grammar rule S -> NP V , V , V ,V , and V. However, since this is a rare occurance in the Wall Street

Journal, there isn't such a rule because such a construction was not in the training data. Similar problems happen for adjective, adverb, and other constructions. However, since the parser always produces a consistent wrong output for such cases, it might be possible to determine a pattern and fix the problem at the Sentence Fragmenter level.

2.6.4 Misparses Due to Incorrect Prepositional Phrase Attachment

Another category of misparses is cases when the prepositional phrase is attached to the incorrect construction. This problem is one that the author of the Apple Pie Parser lists as a known problem area. The problem occurs because without lexical information, it is impossible to determine where the prepositional phrase[9] should be attached. Consider the following example.

```
the commander must exercise constraint when defining the role of deception
within the overall concept of his operations
```

The Apple Pie Parser generated the following incorrect parsed tree.

```
(S (NP (NPL the response)
       (PP of
           (NP the
               (NP (NPL us)
                   service
                   components)
               and
               (NP (NPL republic)
                   (PP of
                       (NP (NPL korea)
                           (PP to
                               (NPL these threat changes)))))))))
   (VP are
       (VP shared
           (PP with
               (NPL each other))))
```

In this example, the prepositional phrase *to these threat changes* is attached to *korea*.

[9]A similar problem exists for conjunctive attachments. However, due to the low frequency of conjunctive attachments in the C2W data, those errors were classified as errors caused due to the use of the incorrect rule.

However, intuition on the meaning of the sentence suggests that the prepositional phrase should be attached to the *the response*.

Prepositional phrase attachment is a classic problem that cannot be resolved by context free grammars. Clearly, without the knowledge of the actual words, we couldn't justify that the second structure is more correct than the first one. However, since we have access to the actual word, it might be able to fix some cases of incorrect attachment in the Sentence Fragmenter.

2.6.5 Misparses Due to Incorrect Fitting

The last category of misparse are the failed attempts at fitted parsing. As explained in section 2.5, the Apple Pie Parser resorts to fitted parsing when it cannot construct the parse tree in any other manner. Out of the 20 sentences that were attempted to be fitted, only 6 of them were correctly fitted. The other 14 sentences resulted in an inaccurate parsing. Since fitted parsing is done as a last resort, there isn't a very easily identifiable way of salvaging the incorrectly fitted sentences. Below is an example of one such misfitted sentence.

```
once targets are selected and c2-attack methods determined, channels must
be established from the source that found the target, to the weapons
system that will attack the target


(S (S (ADVP once)
      (NPL targets)
      (VP are
          (ADJP selected)))
   and
   (S (SS (NPL c2-attack methods)
          (VP determined))
      -COMMA-
      (NPL channels)
      (VP must
          (VP be
              (VP established
                  (PP from
                      (NPL the source)))))))
```

```
(S (NPL that)
   (VP found
       (NPL the target)
       -COMMA-
       (PP to
           (NPL the weapons system))))
(S (NPL that)
   (VP will
       (VP attack
           (NPL the target)))))
```

2.6.6 Misparses Due to Punctuation Errors

Another cause for misparses, which hasn't been mentioned so far is a misparse due to punctuation marks such as ':', '-', '(', or ')'. The author of the Apple Pie Parser recognized this as an area which needs further work. The reason why these punctuation marks cause problems is because these are relatively rare tokens in the training data. Therefore, the statistical parser only knows of a few ways these punctuation marks are used. Also, the parentheses cause further problems because the parser was trained by sentence and some parentheses span over multiple sentence. This causes odd occurances such as unmatched parenthesis. An example where such a punctuation causes a problem is given below:

```
deception cycle: the entire process and procedure for preparing for
planning, executing, and evaluating deception operations

(S (SS (NP (NP (NPL Deception)
               Cycle)
           -COLON-
           (NP (NPL The entire process)
               and
               (NPL procedure)))
       for
       (VP preparing
           (PP for
               (NPL planning))))
   -COMMA-
   (SS (VP executing))
   -COMMA-
   and
   (SS (VP evaluating
           (NPL deception operations))))
```

In the previous example, the colon causes a problem because it is not used as a

sentential barrier like it was supposed to. The previous parse tree seems to suggest that *deception cycle* is only related to *the entire process and procedure.* However, clearly it relates to the whole sentence.

To handle these and other similar problems, the input sentence is run through a pre-processer which "cleans" up the input sentence before it is sent to the Apple Pie Parser. One of the things that the preprocessor does is it replaces colons with periods. For the purposes of sentence fragmentation in the domain we are interested in, experimental data supported that colons can be treated as periods. The preprocessor also removes the second comma in the phrase *apples, oranges, and bananas.* This was also a result of experimental results which generated better performance without the second comma.

2.7 Summary

The Sentence Fragmenter is built on the output of the Apple Pie Parser and therefore understanding the inner working of the Apple Pie Parser is very important. The Apple Pie Parser is a probabilistic bottom-up chart parser which a couple of enhancements. These enhancements include a strategy for factoring active nodes as well as techniques of secondary backup grammar and fitted parsing. The parser uses probability assignments of both the lexical items and grammar rules to assign a score to each possible complete parse tree. Then, it returns the best parse tree among the possibilities which reflects its best guess.

The experimental results show that the Apple Pie Parser is parses completely correctly about 55 percent of the time on sentence from the C2W domain. The errors were studied and classified into five different categories, which are incorrect rule, two non-

terminal grammar side effect, prepositional phrase attachment problem, incorrect fitted parsing, and incorrect part-of-speech tagging. This was done because by understanding the nature and frequency of the errors, it is possible to fix some of them in the Sentence Fragmenter stage.

Chapter 3

3 Sentence Fragmentation

Long complex sentences induce a higher degree of ambiguity than shorter simple sentences. This causes some problems for our existing language understanding system, TINA because the processing load due to ambiguity grows exponentially with sentence length. To solve this problem, we decided to use the output of the previously described Apple Pie Parser[1] to fragment complex sentences into simpler pieces that could be individually translated and then recomposed at the semantic frame level.

This chapter presents an overview of the Sentence Fragmenter. It also presents the description of the algorithm, including methods for dealing with Apple Pie Parser errors. Then, the experimental results of the Sentence Fragmenter on the 200 sentences mentioned from the previous chapter are presented.

3.1 Overview of Sentence Fragmenter

The Sentence Fragmenter is an independent module written in C that takes in the

[1]We used the Apple Pie Parser because it is robust and computationally efficient

bracketed output of the Apple Pie Parser as the input and produces a list of fragments as an output. We decided to make the Fragmenter a separate module rather than modify the existing Apple Pie Parser because it provided better abstraction and minimized the chances of introducing new bugs, therefore improving reliability.

Given the bracketed sentence input, the Fragmenter recreates the tree structure with additional information necessary to easily examine the relationships between nodes. Each node keeps track of their surroundings including a pointer to its parent and its children. These pointers are used extensively in determining the individual fragments.

After the tree structure is created, the Fragmenter processes the tree to fix certain known Apple Pie Parser errors as well as replace some tags with others to facilitate the understanding process. For example, certain category labels can be combined with others because they do not change anything for the purposes of sentence fragmentation. The category label NX is replaced with the label NP. Other changes to the tree include removing unnecessary SS and SBAR labels as determined through experimentation.

Once this step is complete, the Fragmenter maintains a list of current fragments. This list initially contains only one tree, which is the tree that represents the input sentence. The algorithm then looks for new fragments in each of the fragments in the list. When new fragments are found, they are extracted from the original tree and added to the list of fragments. This allows the Fragmenter to find fragments which are imbedded under another fragment. This process is continued until no more sub-fragments can be found. The Fragmenter also keeps track of the index for each type of fragment so that it can assign unique tags for each fragment. When a fragment is found, the Fragmenter leave a unique marker which is used later to recompose the sentence.

The process is easier to see with an example. Consider the sentence, *john who was hungry went to a store to buy a sandwich, a salad, and a soda*. By breaking up the sentence into simpler fragments, we can simplify the analysis of this sentence.

First, this sentence is fed through the Apple Pie Parser, which produces the following output.

```
(S (NP (NPL john) (SBAR (WHNP who) (SS (VP was (ADJP hungry))))) (VP went
(PP to (NPL the store)) (TOINF (VP to (VP buy (NP (NPL a sandwich) -COMMA-
(NPL a salad) -COMMA- and (NPL a soda)))))))
```

From this, the following tree structure is created.

```
(S (NP (NPL john)
       (SBAR (WHNP who)
             (SS (VP was
                     (ADJP hungry)))))
   (VP went
       (PP to
           (NPL the store))
       (TOINF (VP to
                  (VP buy
                      (NP (NPL a sandwich)
                          -COMMA-
                          (NPL a salad)
                          -COMMA-
                          and
                          (NPL a soda)))))))
```

In the tree correcting stage, the -comma- between *a salad* and *and a soda* is removed. In this particular example, that was the only correction necessary. Next, the tree is added to the list of fragments. At this point, the Fragmenter recognizes *who was hungry* as a relative clause, and adds it to the list of fragments which now contains two trees.

```
Main Sentence
(S (NP (NPL john)
       relclause1
   (VP went
       (PP to
           (NPL the store))
       (TOINF (VP to
                  (VP buy
                      (NP (NPL a sandwich)
                          -COMMA-
                          (NPL a salad)
                          and
```

```
                              (NPL a soda)))))))

Relative Clause 1
(SBAR (WHNP who)
      (SS (VP was
          (ADJP hungry)))))
```

Then, the Fragmenter recursively handles the coordination of *a sandwich, a salad, and a soda.* At this point, the list of fragment trees look like:

```
Main Sentence
(S (NP (NPL john)
       relclause1
   (VP went
       (PP to
           (NPL the store))
       (TOINF (VP to
                  (VP buy
                      (NP (NPL a sandwich)
                          -COMMA-
                          ndtopic1))))))

Relative Clause 1
(SBAR (WHNP who)
      (SS (VP was
          (ADJP hungry)))))

And Topic 1
(NP (NPL a salad)
    and
    ndtopic2))

And Topic 2
(NP (NPL a soda))
```

Finally, the Fragmenter recognizes the to-infinitive clause. This modifies the main sentence fragment and adds a new fragment to the list.

```
Main Sentence
(S (NP (NPL john)
       relclause1
   (VP went
       (PP to
           (NPL the store))
       toinfc1))

Toinf Clause 1
(TOINF (VP to
           (VP buy
               (NP (NPL a sandwich)
                   -COMMA-
                   ndtopic1))))))
```

From these five fragment trees, the Fragmenter produces the following output.

```
john relclause1 went to the store toinfc1
ndtop1 a salad and ndtopic2
toinfc1 to buy a sandwich comma ndtopic1
relclause1 who was hungry
ndtop2 a soda
```

The order of the output fragments is not guaranteed other than that the main sentence will be the first fragment. However, this is not important because the unique tags allows the sentence to be fully reconstructed.

3.2 Working with known Apple Pie Parser errors

Some of the Apple Pie Parser errors happen regularly enough that it is possible to determine a pattern. Some of these errors can be corrected at the Fragmenter level because we can use additional data such as context information. Also, we can modify the tree and use it to help in the understanding process as long as it does not have a negative side-effect on the fragmentation.

One common pattern that emerges from the Apple Pie Parser is that when it cannot determine the category of a word, it sometimes uses the tag UCP (uncategorized phrase) to mark those words. However, in most of the examples we have seen, UCPs are imbedded under a NP (noun phrase) and the words within the UCP are also nouns. Therefore, the algorithm simply strips the UCP tag, treating the contents as NP. The following sub-tree is replaced with the new sub-tree.

```
BEFORE:
(NPL (UCP devices and other)
     materials))))

AFTER:
(NPL devices and other materials))))
```

This is a much more accurate result because the word *other* is clearly associated with

materials.

Another group of errors of the Apple Pie Parser can be fixed with the help of lexical information that is available to the Fragmenter. One such problem is the prepositional phrase attachment problem. The prepositional phrase attachment ambiguity can be most easily solved by referring to the lexical informantion. Consider the following sentence.

```
The response of the us service components and republic of korea to these
threat changes are shared with each other
```

The tree structure for this sentence is the following.

```
the commander must exercise constraint when defining the role of deception
within the overall concept of his operations
```

The Apple Pie Parser generated the following incorrect parsed tree.

```
(S (NP (NPL the response)
       (PP of
           (NP the
               (NP (NPL us)
                   service
                   components)
               and
               (NP (NPL republic)
                   (PP of
                       (NP (NPL korea)
                           (PP to
                               (NPL these threat changes)))))))))
   (VP are
       (VP shared
           (PP with
               (NPL each other))))
```

The parse tree above suggests that the prepositional phrase *to these threat changes* is attached to the korea. But it is clear from the lexical property of *response* that the prepositional phrase should be attached to the *response*. How can we fix this problem? The key is the word *response* because it is a noun that often has a prepositional phrase that begins with the word *to* associated with it. Therefore, anytime the word *response* is the noun phrase of a structure which contains a prepositional phrase that begins with the word *to*, it is more likely attached to it. The previous example is converted to the following.

```
(S (NP (NPL the response)
       (PP of
           (NP the
               (NP (NPL us)
                   service
                   components)
               and
               (NP (NPL republic)
                   (PP of
                       (NP (NPL korea))))))
       (PP to
           (NPL these threat changes)))
   (VP are
       (VP shared
           (PP with
               (NPL each other))))
```

Yet another group of errors that the Apple Pie Parser makes are due to the fact that the only non-terminals of the grammar are S and NP. If we can recognize a constant pattern of the errors caused by this, it is possible to fix it at the Fragmenter level. This is true for cases involving coordination of more than three adjectives. Consider the following sentence.

```
areas that can be included are visual, electronic, sonic and olfactory
```

This generates the following output.

```
(S (NP areas
       (SBAR that
             (SS (VP can
                     (VP be
                         (VP included))))))
   (VP are
       (ADJP visual)
       -COMMA-
       (SS (ADJP electronic -COMMA- sonic and olfactory))))
```

Because the grammar of the Apple Pie Parser is based on the non-terminals NP and S, the parser can only handle coordinations of terminals such as adjectives and verbs only if has a grammar rule for them. Since the grammar rules were extracted automatically from the tree-bank, the only way the Apple Pie Parser could parse the above sentence correctly is if there was a sentence with four coordinated adjectives in the tree-bank. Therefore, when faced with such sentences, the Apple Pie Parser treats the latter three as S, because there is

grammar rule which treats the three adjectives as an SS. This is done using grammar rule number 32841 which is shown below.

```
271 : 415 406 415 409 415
 :score 98
  :struct  "(SS (ADJP <1> <2> <3> <4> <5>))"  ;
```

These and other similar problems that were caused because of only two non-terminals can be generally fixed, if they occur regularly, in the Fragmenter. For this particular example, the Fragmenter outputs:

```
areas relclause1 are visual comma ndmod1
ndmod1 electronic comma ndmod2
relclause1 that can be included
ndmod2 sonic and ndmod3
ndmod3 olfactory
```

3.3 Sentence Fragmentation Techniques

The overall strategy of sentence fragmentation is to fragment a sentence into smaller pieces if we can do so while preserving the meaning of the sentence and the fragments. The goal is to reduce the ambiguity of the original sentence by making it shorter. Since we want to preserve the meaning of the main sentence and the fragments, we have to decide boundaries at which both the main sentence and fragments have a meaning of their own. The fragments can be extracted, translated, and re-inserted without much loss of information.

Such fragments include coordinates, relative clauses, complement clauses, adverb clauses, to-infinitive clauses, adverb prepositional phrases, and therefore clauses. The definition and algorithm to determine each type of fragment are discussed in the following sections.

The concept of determining fragments is an important sentence fragmentation

technique because such fragments allow sentences to be divided into smaller pieces without sacrificing the meaning of the original sentence. Often, long sentences, which are the sentences we are really interested in fragmenting, contain several clauses. Thus, determining and fragmenting such clauses is an important part of the Fragmenter.

3.3.1 Definitions of Fragments

Nouns, verbs, adjectives, noun phrases, and verb phrases can all be coordinated by using coordinators such as comma, and, or, but, etc. Coordinates can easily be identified and therefore, they are a good target for fragmentation. By examining the output of the Apple Pie Parser, we can easily determine the cases of coordination. The following is an example of a sentence with a verb phrase coordinate.

```
john bought the house and sold the car
```

Similar to verb phrase coordination and noun coordination, other types of coordination can be recognized and fragmented. The Fragmenter currently handles verb phrase coordination, verb coordination, noun coordination, adjective phrase coordination, and sentence level coordination.

A relative clause is a clause that modifies the noun. In the Fragmenter, it is defined to be a VP or a clause[2] immediately dominated[3] by an NP.

```
the man who is tall walked down the street
```

In the sentence above, the relative clause *who is tall* modifies *the man*. Clearly, extracting *who is tall* does not affect the overall meaning of the sentence. Also, the fragment *who is tall*

---

[2]A clause is defined as a SS or SBAR structure

[3]X immediately dominates Y if and only if Y is nested exactly one step under X.

can be separately translated.

A complement clause is similar to a relative clause in that it has its own meaning. The difference is that while a relative clause is optional within a sentence, a complement clause is obligatory. A complement clause is often but not always preceded by a special word called the complementiser, which marks the beginning of the clause.

```
john thought that the world is flat
```

The complementiser is the word *that* and the complement clause is *that the world is flat*. Due to the complex nature of complement clauses, the algorithm that recognizes complement clauses is more complex than the previous examples.

A complement clause is defined as a clause which is immediately dominated by a VP.. This captures the cases where the actual verb is followed by a complete idea, as in the example given above.

```
military deception can be used to enhance the impact obtained from
practicing these tenets
```

A second definition of a complement clause is a sentence phrase embedded under a prepositional phrase. In the sentence above, *practicing these tenets* is a complement clause.

An adverb clause is a clause that functions like an adverb. The clause typically modifies a verb or verb phrase and is optional in the sentence.

```
if the apple is ripe, ship it to the wholesaler
```

The phrase *if the apple is ripe* is an adverb clause that modifies the verb *ship*. The main sentence is *ship it to the wholesaler*. Adverb clauses are defined in the Fragmenter as a sentence phrase (SBAR) followed by a comma and then a verb phrase.

A to-infinitive clause is similar to a complement clause in the sense that it is not optional but can be seperated from the main sentence without affecting the meaning.

```
john saved money to buy a house
```

The phrase *to buy a house* is the to-infinitive which describes the consequent action of the main sentence. The main sentence is then *john saved money <to do something>*. Clearly, the to-infinitive does not change the meaning of the sentence. To-infinitive are defined to be a verb dominated by the verb *to* in the Fragmenter.

Another clause like structure is the adverb prepositional phrase. An adverb prepositional phrase is a prepositional phrase that modifies the verb.

```
after the deception operation is completed, conduct an evaluation
```

The phrase *after the deception operation is competed* is a prepositional phrase that gives extra information about the main sentence. However, it can be treated as a separate fragment because it contains a unique noun phrase and a verb phrase. The current Fragmenter algorithm classifies a PP structure followed by a comma as an adverb prepositional phrase. This strategy was successfully tested with many examples to make sure this is an accurate classification.

In addition to using the structure of the Apple Pie Parser output as a way of fragmenting sections, it is also possible to make use of the knowledge of the English language to determine certain cases where fragmentation is possible. The two English words that can be used to our advantage are the words *therefore* and the word *so*. These are instances of clause conjunctions. Consequently, the two fragments are named the therefore-clause and the so-clause respectively. Consider the following sentence.

```
a feint must appear real, therefore some contact may be required
```

In this example, we can use our knowledge of the word *therefore* to fragment the sentence into two parts. Words like *therefore, so, hence,* etc. clearly put two sentences

together in the English language. Currently the algorithm only handles the words *therefore* and *so* because those two words occur the most in the domain we are interested in. An improvement that can be implemented in the future could dynamically handle any of those words by looking up the words from an user editable list of such words.

3.3.2 Example of Fragments

The following is an example of verb phrase coordination.

```
john bought the house and sold the car

(S (NPL john)
   (VP (VP bought
           (NPL the house))
       and
       (VP sold
           (NPL the car))))

OUTPUT:

john bought the house and ndvp1
ndvp1 sold the car
```

The following is an example of noun coordination.

```
the fruit store sold apples, oranges, peaches, purple grapes, and large
watermelons

(S (NPL the fruit store)
   (VP sold
       (NP (NPL apples)
           -COMMA-
           (NPL oranges)
           -COMMA-
           (NPL peaches)
           -COMMA-
           (NPL purple grapes)
           -COMMA-
           and
           (NPL large watermelons))))

OUTPUT:

the fruit store sold apples comma ndtopic1
ndtop1 oranges comma ndtopic2
ndtop2 peaches comma ndtopic3
ndtop3 purple grapges and ndtopic4
ndtop4 large watermelons
```

The following is an example of a relative clause.

```
the man who is tall walked down the street

(S (NP (NPL the man)
       (SBAR (WHNP who)
             (SS (VP is
                     (ADJP tall)))))
   (VP walked
       (PP down
           (NPL the street))))

OUTPUT:

the man relclause1 walked down the street
relclause1 who is tall
```

The following is an example of a complement clause.

```
john thought that the world is flat

(S (NPL john)
   (VP thought
       (SBAR that
             (SS (NPL the world)
                 (VP is
                     (ADJP flat))))))

OUTPUT:

john thought compclause1
compclause1 that the world is flat
```

The following is an example of an adverb clause.

```
if the apple is ripe, ship it to the wholesaler

(S (SBAR if
         (SS (NPL the apple)
             (VP is
                 (ADJP ripe))))
   -COMMA-
   (VP ship
       (NP (NPL it)
           (PP to
               (NPL the wholesaler)))))

OUTPUT:

adverbc1 comma ship it to tne wholesaler
adverbc1 if the apple is ripe
```

The following is an example of a to-infinitive clause.

```
john saved money to buy a house

(S (NPL john)
   (VP saved
       (NPL money)
       (TOINF (VP to
                  (VP buy
                      (NPL a house))))))

OUTPUT:

john saved money toinfc1
toinfc1 to buy a house
```

The following is an example of an adverb preposition.

```
after the deception operation is completed, conduct an evaluation

(S (PP after
       (SS (NPL the deception operation)
           (VP is
               (VP completed))))
   -COMMA-
   (VP conduct
       (SS (NPL an evaluation))))

OUPUT:

adverbpp1 comma conduct an evaluation
compclause1 the deception operation is completed
adverbpp1 after compclause1
```

The following is an example of a therefore clause.

```
a feint must appear real, therefore some contact may be required

(S (SS (NPL a feint)
       (VP must
           (VP appear
               (ADJP real))))
   -COMMA-
   (ADVP therefore)
   (SS (NPL some contact)
       (VP may
           (VP be
               (VP required)))))

OUTPUT:

a feint must appear real comma thereforeclause1
thereforeclause1 therefore some contact may be required
```

3.4 Experimental Results

To judge the performance of the Fragmenter, the two hundred sentences used in the previous chapter on the Apple Pie Parser were used as a test set. The sentences were run through the Fragmenter and manually examined to determine the correctness. Through this testing, I wanted to answer several questions regarding the performance of the Fragmenter. Since the algorithms were generally designed with the correct parse as the input, we expect the Fragmenter to be accurate for those sentence for which the Apple Pie Parser parsed correctly. However, how would the Fragmenter handle misparsed sentences? How many fragments would it determine incorrectly? To answer these questions, the output of the Sentence Fragmenter on the 90 misparsed sentences were examined.

Table 3-1 below shows the number of misparses that actually caused fragmentation errors. As stated early, not all incorrectly parsed sentence lead to a fragmentation error. The sentences which are misparsed but doesn't cause a fragmentation error are marked in Appendix B. Misparsing only affects sentence fragmentation if the misparsing leads to an extra fragment or causes the fragment to be missed. Examples of such sentence are given later when the individual types of errors are discussed.

Misparse Type	Misparsed	Fragmentation Error
Incorrect rule	32	23
Two non-terminal side effect	4	3
Incorrect PP attachment	14	8
Incorrect fitting	14	13
Incorrect part-of-speech tagging	26	20
Total	**90**	**67**

Table 3-1 Misparses and Number of Fragmentation Errors

From Table 3-1 we can see that 67/90 or 74% of the misparsed sentence have fragmentation errors. To find out why some misparsed sentences don't cause fragmentation errors and to find out why some types of misparses cause a higher or lower percentage of misfragmentation, we need to look at the relationship between misparses and sentence fragmentation.

Sentence that are misparsed due to the use of an incorrect rule cause a fragmentation error 23/32 or 72% of the time. Many of these misparsed sentence do not cause a fragmentation error if the structure of the sentence is simple. This happens most (but not necessary limited to) with shorter sentences. For example, the sentence:

```
use an implementation schedule at tab d to control complex deception
operations

(S (VP use
       (NPL an implementation schedule)
       (PP at
           (NPL tab))
       (NPL d)
       (TOINF (VP to
                  (VP control
                      (NPL complex deception operations))))))
```

This sentence is clearly misparsed because the correct parse tree should look like the following.

```
(S (VP use
       (NPL an implementation schedule
            (PP at
                (NPL tab d)))
       (TOINF (VP to
                  (VP control
                      (NPL complex deception operations))))))
```

However, the output of the Fragmenter is not affected because the only fragment that the Fragmenter recognizes in either case is the to-infinitive clause. The output is:

```
use an implementation schedule at tab d toinfc1
toinfc1 to control complex deception operations
```

For the misparses caused by the S-NP grammar limitation, one of the four sentence is salvaged by the fragmentation algorithm. The following sentence is clearly misparsed but is corrected by the Fragmenter, producing the correct output.

```
areas that can be included are visual, electronic, sonic and olfactory
```

```
(S (NP areas
       (SBAR that
             (SS (VP can
                     (VP be
                         (VP included))))))
   (VP are
       (ADJP visual)
       -COMMA-
       (SS (ADJP electronic -COMMA- sonic and olfactory))))
```

This sentence is misparsed because of the inability of the Apple Pie Parser to handle the four coordinated adjectives. However, since this pattern is very common, the Fragmenter corrects the following tree to the following.

```
(S (NP areas
       (SBAR that
             (SS (VP can
                     (VP be
                         (VP included))))))
   (VP are
       (ADJP visual -COMMA- electronic -COMMA- sonic and olfactory))))

areas relclause1 are visual comma ndmod1
relclause1 that can be included
ndmod1 electronic comma ndmod2
ndmod2 sonic and ndmod3
ndmod3 olfactory
```

By taking advantage of re-occuring misparse patterns and their causes, It is possible

to overcome shortcomings of the Apple Pie Parser.

The third category of misparses, misparses due to incorrect prepositional phrase attachment, generally does not cause a fragmentation error by itself because prepositional phrases are not considered to be fragments. This explains the relative low (8/14 or 57%) percentage of misparses that cause a fragmentation error. However, when other types of fragmentations occur, the prepositional phrase might be wrongly attached to the wrong fragment. And example of one such sentence is:

```
reject actions that add undo complexity to a deception plan without
materially improving its chances of success

(S (VP reject
       (NP actions
           (SBAR that
                 (SS (VP add
                        (NPL undo complexity)
                        (PP to
                            (NPL a deception plan))))))
       (PP without
           (SS (ADVP materially)
               (VP improving
                   (NP (NPL its chances)
                       (PP of
                           (NPL success)))))))))
```

ACTUAL OUTPUT:

```
reject actions relclause1 without compclause1
relclause1 that add undo complexity to a deception plan
compclause1 materially improving its chances of success
```

CORRECT OUTPUT:

```
reject actions relclause1
relclause1 that add undo complexity to a deception plan without
     compclause1
compclause1 materially improving its chances of success
```

The preposition phrase *without materially improving its chances of success* should be attached to the verb *add* instead of the verb *reject*. The algorithm decides that *that add undo...* is a relative clause. If the preposition phrase was attached properly, it would have been a part of the relative clause rather than part of the main sentence.

This problem is fixed in the Fragmenter for the case where the verb is *response* and

the preposition is *to*. A further enhancement that can be implement in the future would be to extend the system to recognize more verb - preposition pairs which can be used to correct more of the prepositional phrase attachment problem.

Sentences that are misparsed due to incorrect part of speech tagging affects sentence fragmentation 20/26 or 77% of the time. Similar to the case where the incorrect rule was chosen, this type of misparse causes the Fragmenter to behave in an unpredictable manner. Therefore, generally only shorter sentence do not affect sentence fragmentation. An example of a short sentence that was misparsed is:

```
designate security measures

(S (NPL designate security)
   (VP measures))

designate security measures
```

The word *measures* is mistagged as a verb instead of a noun. However because the sentence only has one fragment, the misparse does not affect sentence fragmentation.

Sentence that are misparsed due to incorrect fitting cause more problems than any other category. Among 14 of the sentences, only one of them was fragmented properly. This is partial due to the fact that most fitted sentences are long and complex. Therefore, they tend to have a lot of fragments. Since misparsing affect sentences that have multiple fragments, sentences that are misfitted can seldom be salvaged in whole. However, some of the fragments can be salvaged.

Misparse Type	Number of Sentence	Number of Fragments	Number Salvaged
Incorrect rule	23	72	29
Two non-terminal side effect	3	9	4
Incorrect PP attachment	8	24	14
Incorrect fitting	13	68	35
Incorrect part-of-speech tagging	20	45	12
Total	**67**	**218**	**94**

Table 3-2 Partially Correct Fragmention

Even when the sentence as a whole is incorrectly fragmented, some of the fragments in the sentence are determined correctly. Therefore, it is possible to salvage those fragments. Table 3-2 shows the total number of fragments and the number of fragments salvaged among those sentence that were not fragmented correctly. In the 67 that were not fragmented correctly, 94 out of 218 or 43% of the fragments were still determined properly. Consider the following misparsed sentence which has ten fragments.

```
deception means are the methods, resources and techniques that can be used
to control friendly, physical, technical and administrative actions to
convey or deny information and indicators to the deception target

(S (S (NPL deception means)
      (VP are
          (NPL the methods -COMMA- resources and techniques)
          (SBAR that
                (SS (VP can
                       (VP be
                           (VP used
                               (TOINF (VP to
                                          (VP control
                                              (NPL (ADJP friendly
                                                         -COMMA-
                                                         physical
                                                         -COMMA-
                                                         technical
                                                         and
                                                         administrative)
                                                   actions)
                                              (TOINF (VP to
```

```
                                                        (VP convey)
                                                        ))))))))))))
or
(S (VP deny
       (NPL information and indicators)
       (PP to
           (NPL the deception target)))))
```

This sentence is misfitted because the parser treated the or as a sentential coordinator instead of the correct verb coordinator. However, the rest of the sentence is not affected by this misparse. Out of the 10 fragments, the misparse only affected one of them. Therefore the 9 other fragments can be salvaged.

ACTUAL OUTPUT:

```
deception means are the methods comma ndtopic1 compclause1 or orclause1
ndtop1 resources and ndtopic3
ndmod1 physical comma ndmod2
toinfc1 to control friendly comma ndmod1 actions toinfc2
compclause1 that can be used toinfc1
ndtop2 indicators
ndtop3 techniques
ndmod2 technical and ndmod3
toinfc2 to convey or orvp1 information and ndtopic2 to the deception
      target
orvp1 deny
orclause1 deny ndmod3 administrative
```

CORRECT OUTPUT:

```
deception means are the methods comma ndtopic1 compclause1 or orclause1
orclause1 deny information and ndtopic2 to the deception target
ndtop1 resources and ndtopic3
ndmod1 physical comma ndmod2
toinfc1 to control friendly comma ndmod1 actions toinfc2
compclause1 that can be used toinfc1
ndtop2 indicators
ndtop3 techniques
ndmod2 technical and ndmod3
toinfc2 to convey
ndmod3 administrative
```

3.5 Summary

The Sentence Fragmenter is built on the output of the Apple Pie Parser. The Fragmenter recursively searches the parse tree for clauses and coordinates and returns a list of fragments that it finds. When a clause is found, the Fragmenter leaves a unique marker

which is later used for reconstruction of the semantic frame.

The fragments that the Fragmenter finds are verb phrase coordination, verb coordination, noun coordination, adjective phrase coordination, sentence level coordination, relative clause, complement clause, adverbe clause, to-infinitive clause, adverb prepositional phrases, *so*-clauses, and *therefore* clauses. It also fixes some known and predictable errors of the Apple Pie Parser such as the UCP problem and the prepositional phrase attachment.

The experimental results show that the Sentence Fragmenter correctly fragments 23 out of the 90 misparsed sentences. Even in the 67 misfragmented sentences, about 43 percent of the fragments are still salvaged.

Chapter 4

4 Role of Fragmentation in MT

The overall goal of the machine translation project is to develop a robust, high-performance machine translation system. To achieve these goals, several different enhancements have been made to the original system. The two major architectural changes are the addition of a Part of Speech Tagger as well as the integration of sentence fragmentation.

The Part of Speech Tagger and the Sentence Fragmenter play important roles in our machine translation system. The use of the Part of Speech Tagger allows the system to translate sentence with words that are not registered in the system grammar. The use of the Sentence Fragmenter reduces the length of the sentences that need to be parsed. This reduces ambiguity which results in better performance in terms of speed.

To examine the effects of these enhancements in machine translation, we need to examine the system before and after the enhancements were integrated. This chapter presents a detailed description of the original machine translation system, the system with the Part of Speech Tagger, and finally the system with the Part-of-Speech Tagger and the Sentence Fragmenter both integrated. Regarding the integration of the Part-of-Speech Tagger, refer to

Lee, Weinstein, Seneff, and Tummala [4]. Regarding the semantic frame composition algorithm, which has been proposed and implemented by Stephanie Seneff and Young-Suk Lee, refer to the project report [6]. The last section contains performance evaluations from which we can clearly see the benefits of these enhancements.

4.1 Translation System Overview

```
INPUT SENTENCE: 0819 z unknown contacts replied incorrectly


{c statement
   :time_expression {p numeric_time
                        :topic {q gmt
                                  :name "z" }
                        :pred {p cardinal
                                  :topic "0819" } }

   :topic {q nn_head
             :name "contact"
             :pred {p unknown
                       : global 1 } }

   :subject 1

   :pred {p reply_v
             :mode "past"
             :adverb {p incorrectly } } }
```

Figure 4-1 Input Sentence and Corresponding Semantic Frame

The most important and perhaps the most computationally difficult part in the machine translation system is the language understanding module. The language understanding module takes as the input a given sentence and produces a semantic frame, which is an annotated structure which captures the meaning of the input. Figure 4-1, taken from Reference [6], gives an example of a semantic frame representation of the input

sentence *0819 z unknown contact replied incorrectly.*

The heart of the language understanding module our of system is TINA. It was developed in 1992 by the Spoken Language Systems Group in the Laboratory for Computer Science at MIT, described in Reference [10]. TINA integrates key ideas from context free grammars, Augmented Transition Networks, and the unification concept. It provides a seamless interface between syntactic and semantic analysis, and produces a highly constraining probabilistic language model. TINA is based on a context free grammar augmented with additional features used to add syntactic and semantic constraints. The grammar rules are converted to a shared network structure which combines rules with similar Right Hand Side constructions for a given Left Hand Side element.

A machine translation system relies on a semantic frame to provide meaning representation of a given sentence. This semantic frame representation is also used to provide constraints in terms of permissible syntactic and semantic structures. In many system, the ability to parse as many sentences as possible is often achieved at the expense of many misparses. TINA utilizes mechanisms that were designed to support a seamless interface between syntax and semantics, leading to an efficient mechanism for constraining semantics. Grammar rules encompass both syntactic and semantics structures. At high levels, they describe the syntactic structures while at the lower levels, they describe the semantic structures. TINA uses a very large number of descriptive category labels, which reflects the meaning of the sentence. Therefore, separate semantic rules are not needed. By encoding meaning in the category labels of the parse tree, it is possible to represent semantic restrictions as well as syntactic restrictions in an efficient manner in the same grammar. This also allows the final semantic frame to be directly extracted from the parse tree.

TINA was designed to allow rapid development of the grammar and/or porting of the grammar to a new domain. This allowed us to develop our machine translation system to work with both MUC-II data and C2W data. The grammar of TINA is initially written by hand and then augmented automatically by using a set of training sentences. The design also includes a framework that easily allows to run the system on a wide variety of grammar and input files through the use of a DEFINITIONS file.

The process of acquiring the TINA grammar is a two step procedure that uses a specific set of sentences. The rule set is first built up by parsing the sentences one-by-one manually, adding new rules as necessary. Once this first step is complete, the parse trees are converted automatically into a set of grammar rules used to parse each sentence. The probability assignments is then established in the second pass from the same set of sentences.

4.2 System with Part of Speech Tagging

The biggest shortcoming of TINA was in its inability to handle unknown words. Because it uses semantic rules defined in terms of part-of-speech tagging, unknown words cause some serious problems, which results in the sentence not being parsed. TINA was originally designed to work in a limited domain with a very limited vocabulary. However, the MUC-II and C2W domains has a much greater vocabulary, not all of which can be learned from the training data set.

Parsing without POS: 0819 Z UNKNOWN CONTACT REPLIED ***INCORRECTLY***

Parsing with POS: ***CARDINAL*** *Z* ***ADJECTIVE NOUN*** REPLIED ***ADVERB***

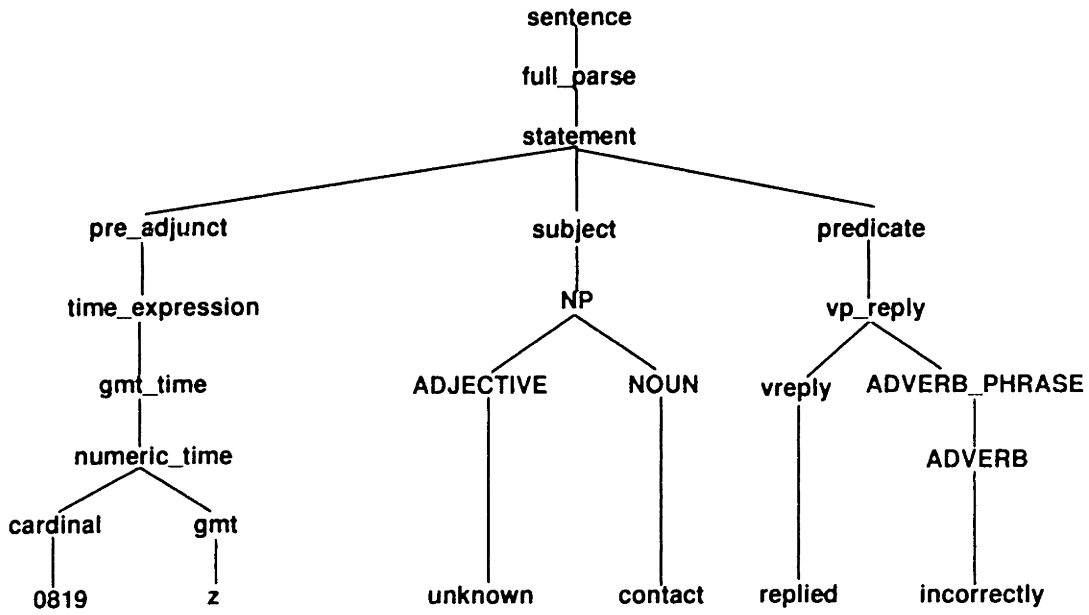

Figure 4-2 Parsing With and Without Part-of-Speech Tagging

In order to handle this problem, TINA was augmented with a Part of Speech tagger. Reference [4], *An Application of Part-of-Speech Tagging To Robust Parsing in Machine Translation of Telegraphic Messages*, describes the part-of-speech tagger that was integrated into TINA. Figure 4-2 from Reference [6], shows an example on how the Part of Speech tagger helps parse the sentence *0819 Z unknown contact replied incorrectly*. The TINA grammar doesn't have a rule for the word *incorrectly*. However, by using the part-of-speech tagger, the original sentence is converted into a structure which has been encountered by the grammar. Specifically, although the grammar doesn't know how to deal with the word incorrectly, it knows how to deal with an adverb following the word *replied*. Therefore by using a Part of Speech tagger, it is possible to write a grammar that captures a larger set of data.

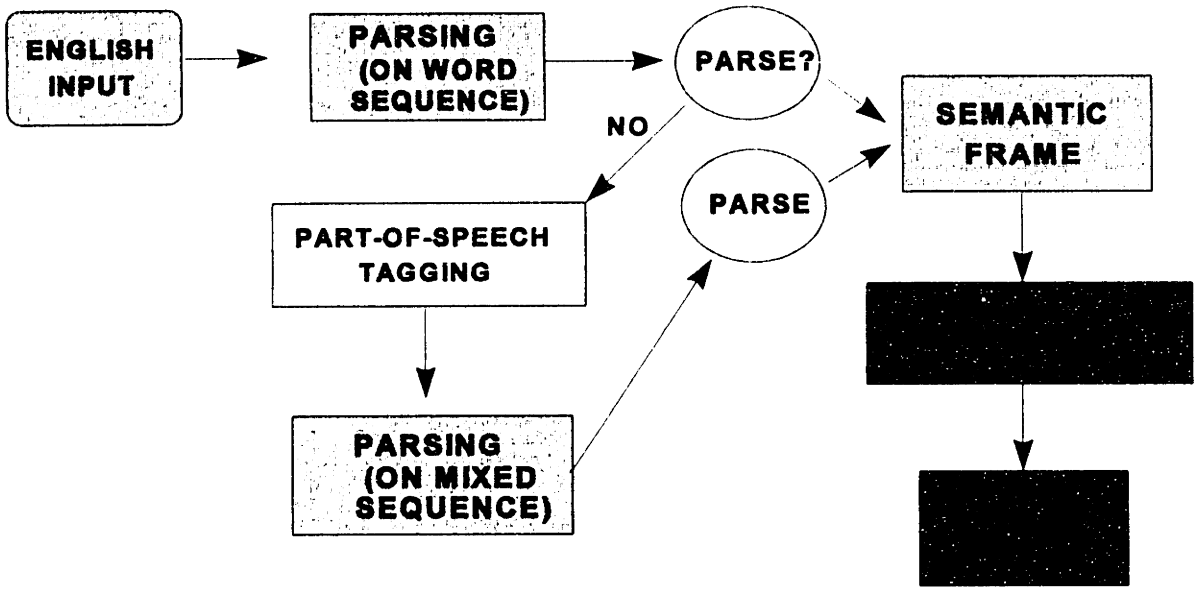

Figure 4-3 Flow Control After the Addition of the Part-of-Speech Tagger

A main design goal in designing the Part of Speech tagger was that it had to integrate seamlessly into the existing system, which minimal side effects. To achieve the goal, a two-step method was implemented where the parser first tries to parse the sentence strictly on the word sequence. If this step is successful, then the system acts exactly as if the Part-of-Speech Tagger did not exist. If this step fails to produce a parse, the system then uses part-of-speech tagging and tries to parse on the mixed sequence. Figure 4-3 from Reference [4], shows the flow of the system. The mixed mode increases the coverage of the grammar without increasing ambiguity. This two-step method guarantees that only sentences that failed to parse in the old system are subjected to the part-of-speech tagging method.

The part-of-speech tagger uses a balance mixed mode of semantic and syntactic information. Rules involving verbs and prepositions are lexicalized to resolve the prepositional phrase attachment ambiguity. Rules involving verbs are also lexicalized to prevent misparsing due to incorrect sub-categorization. Domain specific expressions which occur frequently in phrases with omitted elements are also lexicalized. Any other syntactic

rules are defined in terms of Part-of-Speech. This mixed mode allows the parser to achieve a broader coverage without introducing more ambiguity. The strategy was refined by testing on many test sentences and lexicalizing the grammar rules which caused misparses.

In integrating the Part-of-Speech Tagger, TINA was adapted to accommodate to take as input a mixture of words and parts-of-speech. A rule based part-of-speech tagger was integrated as a preprocessor to the parse. The advantage of using a part-of-speech tagger over a lexicon containing part-of-speech information is that the former can tag words which are new to the system based on the surrounding words. This results in a more robust system that can deal with any unknown word, such as unique acronyms. Once the parse tree is generated using the parts-of-speech, the semantic frame representation contains parts-of-speech as terminals, which is not very desirable for machine translation purposes. Thus, after the parse tree is generated, the parts-of-speech are replaced with the actual input words.

The actual part-of-speech tagger used is the Transformation-Based Part-of-Speech tagger which uses the transformation-based error-driven learning algorithm. This tagger was chosen because it is a rule-base tagger which achieves high performance with a training corpus of modest size, which was well-suited to our use because our training corpus is fairly small. The transformation-based tagger, on the first pass, tags all the words in isolation by using the most likely possibility. Unknown words are assumed to be nouns and then upgraded to the most likely tag by using prefixes, suffixes, infixes, and adjacent words. In the second pass, contextual transformations are used to improve the accuracy. The part of speech tagger achieves a very high performance of 98 percent accuracy and 82 to 87 percent

accuracy on unknown words[4].

4.3 System with POS Tagging and Fragmentation

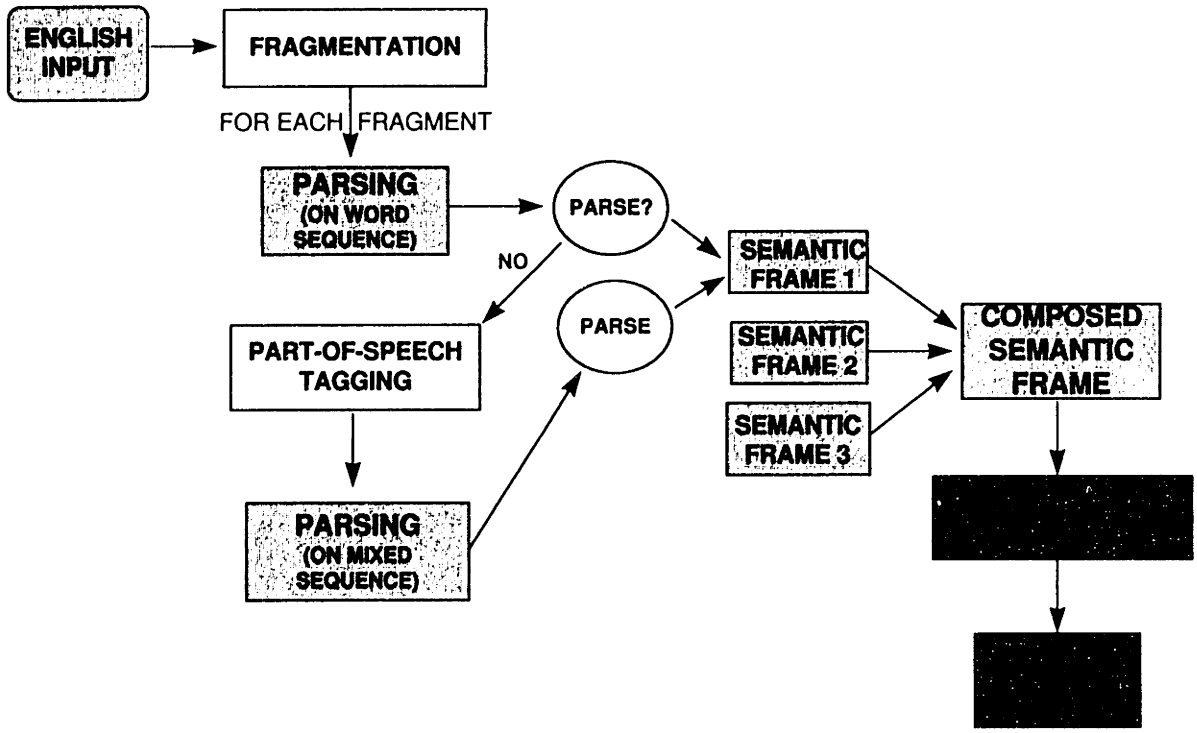

Figure 4-4 Flow Control After the Addition of the Sentence Fragmenter

TINA augmented with the Part-of-Speech tagger is very robust and accurate with short sentences with simple construction. However, when presented with a longer sentence with a complex construction, TINA's grammar cannot cover the constructions. Additionally, longer sentences introduce more ambiguity as well as increase the computational complexity. This problem stems from the fact that TINA's grammar is limited to the constructions of the training sentences.

To overcome this problem, the Sentence Fragmenter was added as a preprocessor to

[4]The results are from Reference [4].

the system. The flow of the parsing processor after the Sentence Fragmenter is integrated is shown in Figure 4-4 from Reference [6]. The input sentence is fed through the Sentence Fragmenter, which generates one or more fragments as the outputs. Then, each of the fragments are fed through the previous parsing strategy, first by just on the sequence of words and if that fails to produce a parse, on the mixed sequence of words and parts-of-speech. Therefore, each fragment generates its own semantic frame. Then, by using the unique tags and labels generated by the Sentence Fragmenter which are accommodated by the grammar, the partial fragments are composed to create a combined semantic frame for the entire input sentence. Then, as before the output sentence is generated by feeding the composed semantic frame through the generation module.

Through the use of the techniques of Sentence Fragmenter, the language understanding module can now handle complex sentences. This additional feature is especially important in the C2W domain where the sentences tend to be very long (20 or more words).

Integrating the Sentence Fragmenter into the existing system involved three major parts. First, the main procedure of TINA had to be modified to call the Sentence Fragmenter module as well as to handle the fragments instead of the original sentence. Second, the TINA grammar had to be augmented to handle the new tags. In addition, TINA had to be augmented to parse fragments instead of complete sentences for which is was designed. Third, the algorithm which composes the semantic frames of the fragments into a combined frame had to be implemented.

TINA was designed to parse one sentence at a time. Therefore, when the Sentence Fragmenter was integrated, we needed to augment the system to work with multiple

fragments for a given sentence. TINA has a main loop that reads a sentence, generates a parse tree, and displays the semantic frame. The modified system has a slightly different flow. It uses the same code to read a sentences but as the next step, it runs the Sentence Fragmenter to determine the number of fragments in the sentence. Then it calls the original parse function for each of the fragments.

The second modification that was necessary was to the grammar. The grammar had to be augmented in two ways. First, it needed to be able to recognize and use the markers left in the original sentence in the grammar. Secondly, TINA needed to be augmented to be able to parse fragments in addition to sentences. These two modifications were made in parallel with the Sentence Fragmenter as new ways of fragmenting were implemented throughout the development cycle.

Once the multiple semantic frames are generated by the parser, we needed an algorithm to recompose the frames into one. The semantic frame composition algorithm, which has been proposed and implemented by Young-Suk Lee and Stephanie Seneff , is described in detail in Reference [6]. By using the markers and labels left in the original sentence and fragments by the Sentence Fragmenter, it is possible to always reconstruct the original sentence or more specifically combine the semantic frames into one. This process is done iteratively as each of the frames are generated by the parser. The frame composition algorithm is run after each fragment is parsed by TINA. Figure 4-6 below shows the flow of the combining process.

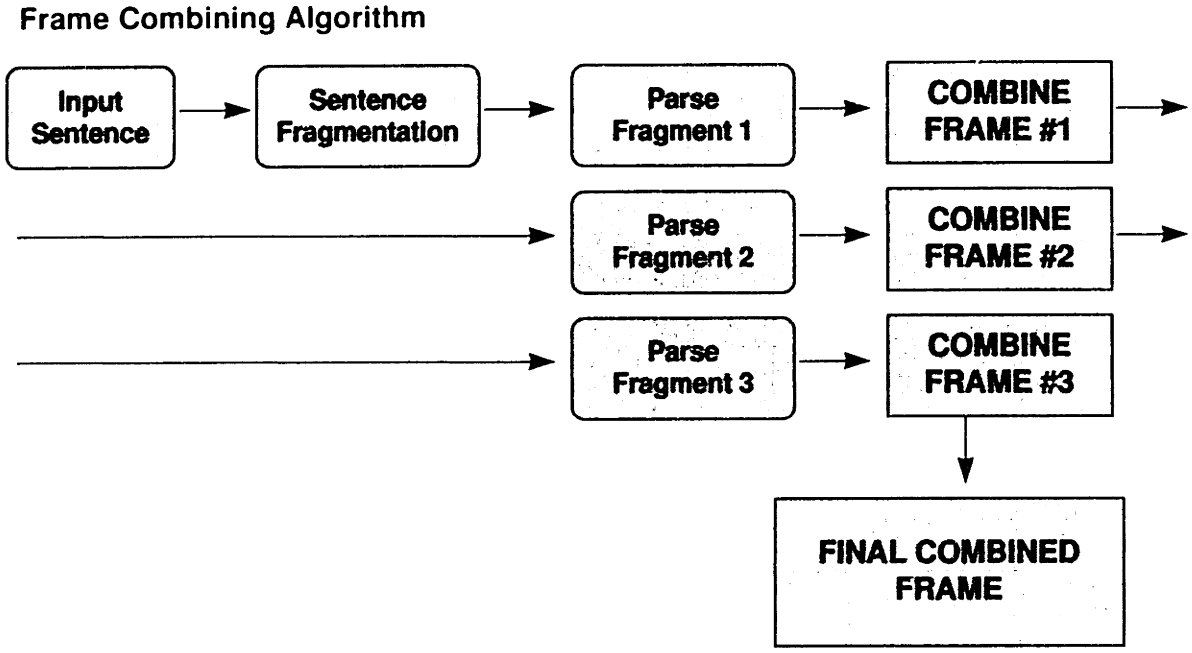

Figure 4-5 Frame Composition Algorithm Flow

The actual frame combination works as follows. The algorithm keeps track of the final combined frame which is initially empty. After the first fragment is parsed, its semantic frame is now stored as the final combined frame. Then as shown in Figure 4-5, the combine frame function is called with the second semantic frame. The combine frame algorithm now takes the label of the second semantic frame and searches for the corresponding marker in the final combined frame. Then, it replaces the node with the marker with the semantic frame of the second fragment. This combining process continues until all the fragments have been parsed and composed.

The following example shows an example of how TINA with the Sentence Fragmenter parses a sentence with multiple fragments.

```
highlight critical elements and vulnerabilities of enemy c2 systems,
operations, and arrangements.
```

This sentence is first put through the Sentence Fragmenter which generates four fragments.

The underlined words have never been seen before in the training data in TINA.

highlight *critical* *elements* and ndtopic1 of enemy c2 systems comma ndtopic2
ndtop2 *operations* and ndtopic3
ndtop3 *arrangements*
ndtop1 *vulnerabilities*

Each four fragments are parsed and results in the following partial semantic frames, listed respectively.

```
{c imperative
   :subject 1
   :pred {p highlight
            :mode "root"
            :topic {q nn_head
                      :name "elements"
                      :and {q ndtopic
                              :name "ndtopic1" }
                      :pred {p critical
                               :global 1 }
                      :pred {p n_of
                               :topic {q nn_head
                                         :name "system"
                                         :number "pl"
                                         :and {q ndtopic
                                                 :name "ndtopic2" }
                                         :pred {p nn_mod
                                                  :topic "enemy c2"
                                                  :global 1 } }
                                      :global 1 } } } }
```

```
{c fragment
   :tag "ndtop2"
   :and {q nn_head
           :name "operations"
           :and {q ndtopic
                   :name "ndtopic3" } } }
```

```
{c fragment
   :tag "ndtop3"
   :and {q nn_head
           :name "arrangements" } }
```

```
{c fragment
   :tag "ndtop1"
   :and {q nn_head
           :name "vulnerabilities" } }
```

By using the frame composition algorithm, these four frames are then combined into one combined frame using the algorithm described earlier in this section.

```
{c imperative
   :subject 1
   :pred {p highlight
            :mode "root"
            :topic {q nn_head
                      :name "elements"
                      :and {q nn_head
                              :name "vulnerabilities" }
                      :pred {p critical
                               :global 1 }
                      :pred {p n_of
                               :topic {q nn_head
                                         :name "system"
                                         :number "pl"
                                         :and {q nn_head
                                                 :name "operations"
                                                 :and {q nn_head
                                                         :name
                                                        "arrangements"
                                                            } }
                                         :pred {p nn_mod
                                                  :topic "enemy c2"
                                                  :global 1 } }
                               :global 1 } } } }
```

4.4 Evaluation Results

To understand the full benefits of the enhancements made to TINA, as always, it is necessary to do some performance evaluations.

CATEGORY	A	B	C	D	E	F	G	Total
Number of sentences	158	26	22	48	24	5	3	286
Number of fragments	283	130	52	165	47	16	3	696
No. of frags per sent	2	4	2.4	4	2.4	3	1	2.4
No. of frags parsed	283	70	27	78	20	6	0	484
Average sent length	7.4	21	15	21	15	20	15	15

A: perfectly parsed and composed
B: improvable by augmenting fragmentation algorithm
C: part-of-speech tagging error
D: apple pie parser error
E: improvable by augmenting grammar
F: improvable by working on TINA
G: sentences whose cause for parsing failure is unclear

Table 4-1 Summary of Performance Results

Table 4-1, extracted from Reference [4], summarizes the performance of the system after all the enhancements have been integrated. From the test results, we can see that 158/286 or 55 percent of the sentences are parsed and composed completely correctly. When we look at partial fragment parsing performance, 484/696 or 70 percent of the fragments are parsed correctly by the system. Clearly, there are still many enhancement that can be made to improve the performance of the system.

4.5 Summary

The Sentence Fragmenter and the part-of-speech tagger play important roles in the machine translation system. The experimental results show that integration of these two enhancements have resulted in a significant improvement over the original system.

The addition of the transformation-based part of speech tagger improves the system's ability to handle unknown words. The use of part-of-speech increases the coverage of the grammar because the grammar can be defined using a mixture of words and parts-of-speech. Therefore, an unknown word does not automatically cause the system to fail.

The addition of the Sentence Fragmenter adds the ability of parsing complex sentences. By dividing complex sentences into smaller fragments, we can reduce the ambiguity and therefore achieve better performance. The fragments are individually parsed and then the semantic frames of the fragments are composed into a combined frame through a frame composition algorithm.

Chapter 5

5 Conclusion

The technique of sentence fragmentation for the purposes of machine translation was the main focus of this thesis. Although the algorithm performs well, there are still many enhancements that can be made to improve the performance of the overall system. In addition, the idea of fragmentation can be extended beyond the field of machine translations into other areas of research. This chapter presents an overview of the thesis, suggests some areas for future improvement to the Sentence Fragmenter, and also presents an example on how the techniques learned from this thesis can be applied in other areas of research.

5.1 Summary of Thesis

The goal of this thesis was to enhance the existing machine translation system to handle complex sentences. The existing system consists of two main modules, one for language understanding and one for language generation. The problem lies in that complex sentences introduce a higher degree of ambiguity, which causes a problem for parsing. To solve the problem, a new separate pre-processor module was designed, implemented, and integrated into TINA.

The basis for the pre-processor module is the output of the Apple Pie Parser. It was used because of its ability to handle long sentences with very good speed performance. The Apple Pie Parser uses a bottom-up chart parsing technique augmented with some strategies for reducing the memory requirements by factoring active nodes with the same label for the left hand side of the grammar rule. It also uses a smaller secondary grammar when it still runs out of memory. In addition to these two techniques for robust parsing, it also uses the idea of a fitted parse when it cannot find a parse tree, especially as a side-effect of using the smaller grammar. Through these three techniques, the Apple Pie Parser manages to efficiently produce a decent output for any arbitrary input. Thus, it makes a good basis to build the Sentence Fragmenter on.

The Sentence Fragmenter module uses the Apple Pie Parser internally. It takes an input sentence, feeds it through the Apple Pie Parser, and uses its output as the basis for the fragmentation algorithm. The Fragmenter converts the output of the Apple Pie Parser into a tree, which facilitates the definition of different fragments by providing a set of relations among the words in the tree. To find all the fragments in the sentences, it recursively searches the tree for the relation definitions. Once the fragments are found, the Sentence Fragmenter leaves a marker in the original sentence for later use and extracts the sub-tree from the main tree. The strategy is repeated to each of the sub-trees until no more new fragments can be found.

The integration of the Sentence Fragmenter into the existing system consisted of many steps. Before the Sentence Fragmenter was integrated, TINA was augmented with a part-of-speech tagger which allows it to use parts-of-speech in addition to individual words in the grammar rather, as described by Lee, Weinstein, Seneff, and Tummala [4]. This allows

the TINA grammar to cover a larger number of sentences. The integration of the Sentence Fragmenter itself consisted of two steps. First, the flow of TINA had to modified such that it could handle more than one fragment per sentences. Second, after each fragment was parsed to produce a partial semantic frame, the partial frames had to be combined to create a combined frame describing the meaning of the whole sentence. This is done through a frame composition algorithm implemented by Stephanie Seneff and Young-Suk Lee [6].

5.2 Future Work

There are a number of future improvements which could lead a to better performance of the Sentence Fragmenter.

Chapter 3 briefly mentioned a couple of improvements that can be made to the Sentence Fragmenter to achieve better performance. The first one involves identifying special word clauses such as *so* clauses and *therefore* clauses. The second improvement that was hinted at was a solution to the prepositional phrase attachment problem.

These two possible future improvements belong to a group of improvements that can be made by making use the lexical knowledge. Because the Apple Pie Parser is based on a context free grammar defined in terms of part of speech, it pays no attention to words. This often leads to misparses which can have been easily avoided had the grammar taken advantage of the lexical knowledge. Improving the Sentence Fragmenter through the use of the actually word is generally something that needs to be identified manually on a case by case basis. However, in some cases, it is possible to generalize the ideas. For example, in the case of the prepositional phrase attachment problem mentioned in chapter 3, it is possible to

build a table of verb - preposition pairs which can be used to reattach the prepositional phrases more appropriately.

Another improvement that can be made to the Sentence Fragmenter is to achieve better performance on MUC-II data. It was mentioned briefly in chapter one that the Sentence Fragmenter research was initially targeted at highly telegraphic MUC-II data. However, partially due to the ungrammatical nature of the MUC-II data, the more grammatical C2W data was used as the basis of the research. However, we are still interested in extending the Sentence Fragmenter to work with MUC-II data.

The main problem with fragmenting MUC-II data is that the ungrammatical nature of the data leads into very poor performance of the Apple Pie Parser. The reason behind this is the fact that the Apple Pie Parser grammar was automatically extracted from the Penn Tree Bank, which is a tagged corpus of sentences from the Wall Street Journal. Therefore, what the parser is really doing is trying to match up test sentence with a previously seen construction. However, since the constuction of sentences are very different between MUC-II and the Wall Street Journal, this is not a very good method.

One possible solution to this problem would be to train the Apple Pie Parser with some manually tagged MUC-II corpus. By somehow combining the existing grammar of the Apple Pie Parser with a new grammar generated from the manually tagged MUC-II corpus, it could be possible to improve performance on MUC-II data while not losing the robustness of the Apple Pie Parser. However, the details on how the two grammars should be combined is unclear and it can only be answered by experimentation. Another major problem that needs to be overcome is that there doesn't exist a tagged MUC-II corpus. Creating any kind of corpus involves at least some human labor and hence is very time intensive.

A third possible improvement would be to improve the performance of the internal part-of-speech tagger of the Apple Pie Parser. The Apple Pie Parser currently uses a simple strategy to assign tags to the words. It does not consider the neighboring words to help determine the part of speech. For example, the word *can* can be used either as a noun or a modal verb. The Apple Pie Parser simply looks this word up in the dictionary and assigns a 1077/1083 or 99.45% score on the modal verb interpretation and only a 6/1083 or 0.55% score of *can* as a noun. Clearly, in cases where the word *the* precedes the word *can*, it is being used as a noun. The Apple Pie Parser part-of-speech tagger has no provision for this and thus still assigns a 0.55% probability of can being used as a noun. The result is that nearly 100% of sentences that use the word can as a noun are misparsed by the Apple Pie Parser.

A solution to this problem would be to replace the Apple Pie Parser's part-of-speech tagger with a more advanced tagger. A good candidate for this would be the same transformation-based part-of-speech tagger that was integrated into TINA. This would probably fix many of the 26 sentences that were misparsed due to wrong part-of-speech tagging. Replacing the Apple Pie Parser's part-of-speech tagger is not a trivial task because it would require reworking some of the internal details of the Apple Pie Parser. In addition, the new tagger has to use the same tags as the current tagger for it to be integrated seemlessly with the existing grammar. However, the rewards could be well worth the effort.

5.3 Applications of Fragmentation Techniques

The techniques used in the Sentence Fragmenter can be applied in other areas of

research. One such possible area of research that this could be used in is information retrieval systems. Current search engines only use a simple string search algorithm on a pre-compiled index to find matching results. This is, for the most part, useful and very fast. However, as computers get faster and faster, there will be enough processing power to allow more complex searching algorithms involving semantics.

To demonstrate a possible use of the Sentence Fragmenter techniques in building complex information retrieval algorithms, an experimental system called NP Finder was implemented from the Sentence Fragmenter through a few minor modifications. The NP Finder takes a sentence as an input and extracts all the noun phrases in the sentence.

```
the big dog ran from the cat
```

The previous sentence returns the following output.

```
Main Sentence:
NP1 ran from NP2

Noun Phrases:
NP1 the big dog
NP2 the cat
```

To see how the NP Finder can be used in information retrieval systems, it is necessary to understand what the problem is with current technologies. Through a clever use of indexes, search engines such as Digital's Altavista are very fast. The main problem is that the search returns many inaccurate matches. Suppose we wanted to find information on "big dog". This returns over 1000 matches on Altavista, which offsets the value of using a search engine to find specific ideas. It returns inaccurate matches including *big dog toys* and *description of the book "little big dog"*.

Returning too many matches is a very common problem when using search engines and as the databases get larger, more matches will occur, compounding the problem. By

using the NP Finder output example shown above, we can design a new information retrieval system which only returns full NP matches. This would prevent the system from matching *big dog toys* and *description of the book "little big dog"*.

This is only one aspect on how the Sentence Fragmenter techniques could be used. It is possible to extend this idea to verbs and other categorys. Furthermore, more interesting search system could be created by allowing the definition of relationships between words. By taking this one step further by adding some semantic information, it could be possible to create an information retrieval system that would allow the user to query for <*something*> *crashing into* <*something*> where <*something*> are types of vehicles, which would match *blue honda*, *big truck*, *a train*, etc.

5.4 Summary

This chapter included an overview of the Thesis, including TINA, the Apple Pie Parser, and the Sentence Fragmenter. Following the overview, several possible strategies for improving the system in the future are presented. Finally, a possible application of the sentence fragmentation technique outside of machine translation systems is discussed.

Appendix A

A Glossary of Linguistic Tags

The list of linguistic tags used in this thesis was compiled from Reference [1].

CC	coordinating conjunction
DT	determiner
NN	noun, singular or mass
NNS	proper noun, singular
NNX	combined tag of NN, and NNS
NP	noun phrase
PP	prepositional phrase
PRP	personal pronoun
S	sentence
VB	verb, base form
VBD	verb, past tense
VBN	verb, past participle
VBP	verb, non-3s, present
VBZ	verb, 3s, present
VBX	combined tag of VBP, VBZ, VBD
VP	verb phrase
UCP	uncategorized phrase

Appendix B

B Test Sentences

These 200 test sentences used for evaluation of the system at different levels. The sentences were randomly selected from chapters 1,3,8, and 11 of the Command and Control Warfare (C2W) handbook. Sentences that were incorrectly parsed by the Apple Pie Parser, yet which did not affect sentence fragmentation are marked with an asterisks (*).

Sentence correctly parsed by the Apple Pie Parser

military deception is a fundamental aspect of the greater whole of military art

military art focuses on the direct use of military force to impose one's will on an opponent

the capability and opportunity to exercise military deception is highly situation-dependent

although opportunities to use deception should not be overlooked, the commander must also recognize situations where deception is not appropriate

military deception can be used to enhance the impact obtained from practicing these tenets, but military deception also provides opportunities to apply these tenets in ways that are not otherwise available

while the role of military deception is relatively simple to understand, its actual use in military operations is not

military deception is functionally related to the broader concept of (c2w and must be applied in that framework

c2w involves all measures aimed at affecting the c2 of the opponent

military deception should be viewed as an important aspect of the c2w effort

the following references provide additional guidance in the preparation and execution of deception

an examination of the common characteristics of successful military deception operations provides a sound

basis for acquiring a practical understanding of the key requirements for success
six common characteristics that underpin successful deception operations are particularly notable
any successful deception operation must provide the target with a wide range of information from multiple sources to form a solid foundation for the deception
effective deception operations usually aim at obtaining a very clear and concise result
the goal and objective of the deception should be easily understood by the originator
comprehensive preparation and exact timing are hallmark characteristics of deception planning and execution
deception is a precision art
the test of value for a deception operation is measured by its success
success is gained if the deception goal is a fully integrated element of the commander's overall concept of operations
such methods usually entail breaches in opsec to convey misleading indicators
common characteristics (paragraph 11-3) are used to reinforce and lead credence to a deception story
a feint must appear real, therefore some contact with the enemy may be required
maintain a staff component within the c2w cell to act as the single working level point of contact for supervision and coordination of deception
tactical deception planners in units, corps and below, will coordinate with lateral units to ensure deception operations do not interfere with other units oplans
consider deception planning while basic operation plans are being developed
this will ensure the plan properly supports and complements operational objectives
the deception plan must influence the deception target
the deception target is ultimately the enemy commander who has the authority to make decisions that will force favorable enemy reaction to the deception objective
the deception story supports the deception objective
the deception story tells the enemy of a believable friendly capability or intent
the story must be realistic and plausible
the deception story must be within our tactical capability as the enemy knows it
it should not project activities that violate our doctrine
planning the following into the deception story is instrumental in its successful execution
information will be verified through the collection of more than one source
the deception story must be consistent with current tactical doctrine
ensure close coordination and control
limit the number of people knowledgeable of the deception
even tasked units may not be cognizant that they are performing a deception operation
there are several different methods for formulating deception plans
during any deception planning, use the deception checklist, tab a, for self-evaluation/appraisal of the plan

this method is not recommended

unwritten deception planning simply executes deception operations as the situation develops

this method has the potential of severe repercussions and should be used only as a last resort

implementation schedules at tab d, are essential for orchestrating and executing the deception in an orderly manner

ensure feedback is incorporated in the plan

during deception plan preparations, continuously review an estimate of the situation and react to changing conditions

in addition to procuring any needed deception devices, other preliminary actions must be completed to include

identify unit to conduct the deception

identify means to command and control the activity

establish logistic support for devices and other materials

it is a scenario for presenting the deception story to the enemy

if the reaction is unfavorable, modify or terminate the operation

this record keeps the commander aware of operational developments, deception execution status and aids in evaluating the deception's success

after the deception operation is completed, conduct an evaluation

use operational and intelligence feedback to evaluate the deception's success

c2-attack encompasses those measures taken to deny enemy commanders and decisionmakers the ability to effectively command and control their forces

active measures are most commonly thought of in the development of a c2-attack strategy

planners should not lose sight of the fact that such a strategy should be supported by the total integration of both active and passive measures

offensive measures include the application of both lethal and nonlethal ea activities

lethal capabilities are targeted against key c2 personnel and their supporting communications systems or facilities

jamming is targeted against those electromagnetic communications and sensors essential to the enemy's c2 in the near term

electronic deception, as another component of ea, conveys false or misleading information to enemy decisionmakers to influence their perception of an operational situation incorrectly and respond, or fail to respond, in a way advantageous to friendly operations

this involves the employment of c2-protect (see chapter 5) in support of c2-attack and illustrates why neither the c2-attack nor the c2-protect strategy can be developed in isolation

also, in a passive sense, intelligence and electronic warfare support (es) are essential in the analysis of enemy c2 and in planning, execution and feedback of c2-attack actions

the following list of unclassified references provides additional useful information about c2-attack planning

and employment

dissuade the enemy from taking actions that would adversely affect mission accomplishment

promote operational superiority at the time and place of engagements

although the authority to develop and implement an effective and integrated c2-attack strategy is delegated to his staff, responsibility for c2-attack remains with the cinc

the c2-attack planner is responsible for coordinating the development of the c2-attack plan with component/supporting force staff elements and translating the strategy into appendix 10 to annex c for cinc unc/cfc/usfk cj3 approval

various staff elements perform duties and functions to support the cj3 and the c2-attack planner in meeting overall c2-attack responsibilities

the following planners are responsible to the cj3 for the integration of their disciplines into the c2-attack strategy

the application of c2-attack against enemy c2 is a complex problem

the process of selecting and identifying c2 targets is difficult because of the large numbers and, in some cases, lack of unique identifiable characteristics

despite the difficulty, all of the following considerations should be examined in developing the overall c2-attack plan

mission accomplishment is the primary consideration in developing a c2-attack plan

regardless of the type of conflict, c2-attack must be considered in the development of the oplan

the threat estimate consists of area of operations, electronic order of battle, c2 activities supporting enemy decisionmakers and weapons systems and estimated enemy intentions

consideration must be given to the possibility that c2-attack actions may affect our ability to collect information from a lucrative source

disruption of enemy command links should be evaluated in terms of the intelligence loss that may result

assets available to execute c2-attack options include forces assigned

such factors include rain, fog, clouds, terrain, propagation conditions and sea state (both open and coastal) which affect radar, transmitters/receivers, jammers and sensors

the decision to employ physical destruction must consider the effects of diverting destructive combat power from other targets

the expendable nature of lethal weapons and their requirement for extremely accurate target location information, could make nonlethal actions more cost effective alternatives

the mobility or location of some c2 targets also makes the consideration of nonlethal counteractions desirable

finally, jamming and deception options may have the potential of affecting several targets simultaneously

destruction, ea, psyop and deception operations must be timed with a friendly operation/mission that can exploit the loss or degradation of enemy c2

however, it could have an adverse impact on friendly operations if it reveals plans and intentions

planners must clearly understand potential enemy c2 targets and their importance to the enemy in both offensive and defensive operations

information in this handbook applies to united nations command , component commands, field armies and other subordinate and supporting organizations, regardless of service

the c2w branch, cjg3 plans division is the unc and cfc/usfk/ground component command for all c2w matters

the organization of the c2w branch, which includes both united states military of all services is shown at figure 1-1

jamming and electronic deception operations, normally referred to as electronic attack , are certainly two essential components of the c2w strategy

formats for appendices which are presented in this handbook are taken from the unc and cfc planning guide and jopes volumes i and ii

c2-attack encompasses those measures taken to deny enemy commanders and decisionmakers the ability to effectively command and control their forces

planners should not lose sight of the fact that such a strategy should be supported by the total integration of both active and passive measures

offensive measures include the application of both lethal and nonlethal ea activities

lethal capabilities are targeted against key c2 personnel and their supporting communications systems or facilities

electronic deception, as another component of ea, conveys false or misleading information to enemy decisionmakers to influence their perception of an operational situation incorrectly and respond, or fail to respond, in a way advantageous to friendly operations

this involves the employment of c2-protect in support of c2-attack and illustrates why neither the c2-attack nor the c2-protect strategy can be developed in isolation

also, in a passive sense, intelligence and electronic warfare support are essential in the analysis of enemy c2 and in planning, execution and feedback of c2-attack actions

the following list of unclassified references provides additional useful information about c2-attack planning and employment

although the authority to develop and implement an effective and integrated c2-attack strategy is delegated to his staff, responsibility for c2-attack remains with the cinc

maximum coordination between c2-attack planning and intelligence and es and communications support activities are established and maintained

a c2-attack plan is issued to all component and supporting force commands that contains sufficient detail to assure maximum guidance and direction in support of mission accomplishment

various staff elements perform duties and functions to support the cj3 and the c2-attack planner in meeting overall c2-attack responsibilities

the following planners are responsible to the cj3 for the integration of their disciplines into the c2-attack

strategy

the application of c2-attack against enemy c2 is a complex problem

the process of selecting and identifying c2 targets is difficult because of the large numbers and, in some cases, lack of unique identifiable characteristics

despite the difficulty, all of the following considerations should be examined in developing the overall c2-attack plan

Sentences incorrectly parsed by the Apple Pie Parser

Due to using the incorrect rule

often, the skillful application of tenets of military operations-initiative, agility, depth, synchronization and versatility, combined with effective opsec, will suffice in dominating the actions of the opponent

*military deception is more subtle, relying on the manipulation of selected appreciations to influence the actions of the opponent

*other functional components of c2w (psyop, opsec, ew and physical destruction) involve more overt measures aimed toward denial, disruption and destruction

*consequently, to achieve its desired effect, military deception operations usually require access to a functional enemy c2 system

key bits of information essential to the deception story may be simply overlooked by the recipient if they are transmitted by a single source

the quality of information is as important as its quantity, including mixing sufficient truth with the falsehoods or distortions to increase their likelihood of acceptance

it is a show of force to gain enemy response, with the friendly force withdrawing without engagement

the net effect of this information must directly lead to an appreciation that is more believable than the reality it is intended to disguise

*included are such areas as sigint, humint and imint which verify desired responses to the deception plan

*the successful deception story must end up in the enemy commander's hands as his staff's estimate of friendly capabilities and intentions

*use an implementation schedule at tab d to control complex deception operations

*the deception story is the information provided to the enemy commander which leads him to an incorrect assessment

*c2-attack planner serves as the key coordinator and point-of-contact within command on all c2-attack matters

at field army and below the cjg3 is the focal point for c2-attack planning and execution

there must be feedback to determine the results of the c2-attack activity for adjusting current or future actions or tactics, or for terminating current actions if the objectives have been accomplished

this forces the enemy to make a decision which puts him at a disadvantage

it is never advisable to undertake a deception that is clearly not within one's technical means and capability

experience has shown that deception plans which do not account for all of the factors necessary to credibly portray the illusion result in failure

the deception concept and the resulting deception story should generally conform to the enemy's preconceptions of how his opponent is expected to behave

the individual whom we are actually targeting may not be the highest ranking officer in the enemy order of battle

the deception objective is what we want the enemy to do or not to do

when developing a deception objective, determine what conclusions the enemy should draw and what actions they should or should not take

sometimes achieving the deception objective is accomplished if the story merely lengthens the enemy commander's decision cycle, resulting in delayed decisions

the indicators displayed must be believable or they will be readily discounted

appendix 7 to annex c, tab c, is used for deception planning to support an oplan, conplan, or an explan

evaluation is an ongoing process as the success of a future operation may depend on the lessons learned from completed operations

determine if the deception was conducted as planned, whether the enemy accepted the deception as real and whether the deception influenced the enemy as desired

*c2-attack planner serves as the key coordinator and point-of-contact within command on all c2-attack matters

c2 can often be reconstructed or redirected to bypass a particular disabled command post, terminal or communications link

it is mutually supported by intelligence to deny information, influence, degrade, or destroy adversary command and control capabilities, while protecting friendly c2 against such actions

these included actions taken to destroy and and or disrupt the functioning of an enemy's c2 capabilities by direct attack against his c2

plan for the integration and execution of those actions designed to deny or convey misleading information to the enemy

Due to part of speech tagging

publish and distribute deception plans and schedules

*designate security measures

such characteristics, as listed below, can provide a general frame of reference for evaluating the relative merits of alternative approaches to specific deception operations

isolated sources of information alone may not be believed

conversely, the deception plan must also conform to the initiator's mission and operational concept

as deception is conducted, measure its effectiveness in terms of enemy reaction

*keep a record of events, documenting all activity

he coordinates c2w disciplines to effectively coordinate and plan for the total integration for these functional areas in support of the overall c2-attack plan

publish and distribute deception plans and schedules

the c2w intelligence support officer is located in the cj2 plans and targets branch and coordinates intelligence support to c2w branch

however, the technical measures that must actually be taken to portray a false reality are normally very complex

improperly timed c2-attack may have little or no influence on the success or failure of the enemy operation

*additionally, the c2w branch is available to coordinate specific training needs as required

this forces the enemy to make a decision which puts him at a disadvantage

*designate security plans

the plan details and formalizes the complete deception operation and, when approved, is the basis for initiating execution on order

the simulate weapons and installations, disguise the appearance of an object, portray the existence of a notional unit, or indicate a different type unit than actually exists

the use of a discarded basic operation plan can be easily modified to support a deception operation

*plan on using multi-spectral operations

conduct deception using deception measures and deception tasks which are directed in the deception plans

this handbook was designed to provide both operators and planners the information needed to successfully plan and execute a command and control warfare strategy

it is organized as a reference guide covering all aspects of c2w in korea including electronic warfare , physical destruction and intelligence support

users are encouraged to extract, amplify and provide useful portions to their subordinate units

*jamming is targeted against those electromagnetic communications and sensors essential to the enemy's c2 in the near term

he coordinates c2w disciplines to effectively coordinate and plan for the total integration for these functional areas in support of the overall c2-attack plan

the c2w intelligence support officer is located in the cj2 plans and targets branch and coordinates intelligence support to c2w branch

Due to two non-terminals S and NP

a mastery of military art is a prerequisite to successful practice of military deception but the mastery of military deception takes military art to a higher level

back of envelope approach is used for time critical situations, a quick method of scratching out a few notes (on the back of an envelope) and then executing the deception

*areas that can be included are visual, electronic, sonic and olfactory

the tactical commander directing the deception should set up a small deception cell to develop, coordinate, monitor and implement the deception plan

Due to incorrect fitting

c2w also involves protective measures taken to protect one's own c2 against similar effort of the opponent, making the overall goal of c2w to obtain a decisive advantage in the ability to manage forces

*without an integrated approach to planning military deception as a component part of c2w, the commander may find himself making choices that unintentionally degrade one capability at the expense of another, rather than synchronizing the total contribution of all aspects of c2w

since it is impossible to totally control all of the information available to the target, deception may involve less of an effort to conceal reality than it does an effort to corrupt the target's perception of reality in a very methodical and convincing manner

even when the material capability exists, if little is done in advance to train and prepare the command to execute deception operations, subsequent attempts at deception may often result in a waste of effort and resources

to be believable, the deceiver is asking his target to accept must be in line with the target's understanding of the originator's warfighting style, military doctrine and battlefield behavior in the existing circumstances

deception means are the methods, resources and techniques that can be used to control friendly, physical, technical and administrative actions to convey or deny information and indicators to the deception target

planning process steps at tab b, is an informal style that can be used in development of a deception plan when an appendix c-7 is not written, or as a basic outline in the development of a deception story

close coordination among all participating services throughout all phases of planning and execution is essential to insure an implemented c2-attack option by one friendly force does not adversely affect the c2 of another friendly force unit

attempts to counter every enemy c2 activity is unrealistic and will prove futile due to the sheer number involved and, in some instances, the value of some of the enemy c2 activities as sources of intelligence

once targets are selected and c2-attack methods determined, channels must be established from the source that found the target, to the weapons system that will attack the target

command and control warfare training is coordinated and conducted throughout the year in each of the functional areas, to keep c2w planners and operators informed of changing roles and requirements

in other words, c2w is a methodical approach to the integrated, timely, balanced and complementary

employment of available lethal and nonlethal means to attack the enemy's c2, while simultaneously protecting our own c2 from similar enemy activities

although each of the elements can be used individually, the maximum effectiveness of c2w is attained only by thoroughly integrating hard kill options, psyop and opsec techniques with ew and deception operations to achieve the synergistic effect of all five elements of c2w

the c2-attack planner is responsible for coordinating the development of the c2-attack plan with component and supporting force staff elements and translating the strategy into appendix 10 to annex c for cinc unc/cfc/usfk cj3 approval

Due to PP-attachment problem

*it is capable of causing the enemy commander to mismanage his forces, leaving him unaware of how his c2 system is being undermined

*military deception differs from other c2w components by virtue of achieving its goals through the orchestrated release of information to the adversary

*the unusual requirement for conduits to the adversary may be at cross-purposes with other functional components of c2w

it may be the one who holds great influence in the decision making process, such as the intelligence or operations officer

reject actions that add undo complexity to a deception plan without materially improving its chances of success

these include opsec actions taken to deny or conceal information from enemy surveillance systems regarding friendly intentions and capabilities

the commander must exercise constraint when defining the role of deception within the overall concept of his operations

deception methods are the specific actions employed to convey elements of the deception story to the target

it may be a simple tactical trick conducted by a lone soldier on a battlefield, or an elaborate stratagem conducted at the theater level

these included actions taken to destroy and disrupt the functioning of an enemy's c2 capabilities by direct attack against his c2

these include opsec actions taken to deny or conceal information from enemy surveillance systems regarding friendly intentions and capabilities

such support would normally be over existing communications channels with no requirement for dedicated c2-attack circuits

*c2-attack planning should be centralized at the unc/cfc/usfk level, with execution at the level exercising operational control over the assets being employed

*such support would normally be over existing communications channels with no requirement for dedicated c2-attack circuits

Appendix C

C Sample Chart Parsing Results

```
the dog chased the cat

---------- chart ------------------
[  0,  1] :    0(  0)  -{   -1}-> =S=
[  1,  2] :   DT(  0)  -{   -1}-> the
[  1,  2] :  NPL( 49)  -{37495}-> DT[1 2]
[  1,  2] :   SS(103)  -{25363}-> NPL[1 2]
[  1,  2] :   NP( 90)  -{17837}-> NPL[1 2]
[  1,  2] :    S(135)  -{ 3259}-> NPL[1 2]
[  1,  3] :  NPL( 20)  -{38966}-> DT[1 2] NNX[2 3]
[  1,  3] :   NP( 61)  -{17837}-> NPL[1 3]
[  1,  3] :   SS( 74)  -{25363}-> NPL[1 3]
[  1,  3] :    S(106)  -{ 3259}-> NPL[1 3]
[  1,  4] :  NPL( 96)  -{39163}-> DT[1 2] NNX[2 3] VBX[3 4]
[  1,  4] :   SS( 58)  -{28334}-> NPL[1 3] VBX[3 4]
[  1,  4] :   NP( 97)  -{20757}-> NPL[1 3] VBN[3 4]
[  1,  4] :    S( 88)  -{ 5358}-> NPL[1 3] VBX[3 4]
[  1,  6] :   SS( 79)  -{28580}-> NPL[1 3] VBX[3 4] NPL[4 6]
[  1,  6] :   NP(114)  -{21318}-> NPL[1 3] VBX[3 4] NPL[4 6]
[  1,  6] :    S( 91)  -{ 5748}-> NPL[1 3] VBX[3 4] NPL[4 6]
[  2,  3] :  VBX( 25)  -{   -1}-> dog
[  2,  3] :   VB( 25)  -{   -1}-> dog
[  2,  3] :  NNX(  1)  -{   -1}-> dog
[  2,  3] :  NPL( 22)  -{41257}-> NNX[2 3]
[  2,  3] :   SS( 76)  -{25363}-> NPL[2 3]
[  2,  3] :   NP( 63)  -{17837}-> NPL[2 3]
[  2,  3] :    S(108)  -{ 3259}-> NPL[2 3]
[  2,  4] :  NPL(109)  -{41467}-> NNX[2 3] VBX[3 4]
[  2,  4] :   SS( 60)  -{28334}-> NPL[2 3] VBX[3 4]
[  2,  4] :   NP( 99)  -{20757}-> NPL[2 3] VBN[3 4]
[  2,  4] :    S( 90)  -{ 5358}-> NPL[2 3] VBX[3 4]
[  2,  6] :   SS( 81)  -{28580}-> NPL[2 3] VBX[3 4] NPL[4 6]
[  2,  6] :   NP(116)  -{21318}-> NPL[2 3] VBX[3 4] NPL[4 6]
[  2,  6] :    S( 93)  -{ 5748}-> NPL[2 3] VBX[3 4] NPL[4 6]
[  3,  4] :  VBN( 10)  -{   -1}-> chased
[  3,  4] :  VBX(  4)  -{   -1}-> chased
[  3,  4] :  NPL( 96)  -{41473}-> VBX[3 4]
[  3,  4] :   SS( 73)  -{35156}-> VBN[3 4]
[  3,  4] :    S(100)  -{15968}-> VBX[3 4]
```

```
[  3,  4] :   NP(164)  -{17824}-> SS[3 4]
[  3,  6] :   SS( 67)  -{35526}-> VBX[3 4] NPL[4 6]
[  3,  6] :   NP(131)  -{22827}-> VBX[3 4] NPL[4 6]
[  3,  6] :    S( 92)  -{15975}-> VBX[3 4] NPL[4 6]
[  4,  5] :   DT(  0)  -{   -1}-> the
[  4,  5] :  NPL( 49)  -{37495}-> DT[4 5]
[  4,  5] :   SS(103)  -{25363}-> NPL[4 5]
[  4,  5] :   NP( 90)  -{17837}-> NPL[4 5]
[  4,  5] :    S(135)  -{ 3259}-> NPL[4 5]
[  4,  6] :  NPL( 19)  -{38966}-> DT[4 5] NNX[5 6]
[  4,  6] :   NP( 60)  -{17837}-> NPL[4 6]
[  4,  6] :   SS( 73)  -{25363}-> NPL[4 6]
[  4,  6] :    S(105)  -{ 3259}-> NPL[4 6]
[  5,  6] :  NNX(  0)  -{   -1}-> cat
[  5,  6] :  NPL( 21)  -{41257}-> NNX[5 6]
[  5,  6] :   SS( 75)  -{25363}-> NPL[5 6]
[  5,  6] :   NP( 62)  -{17837}-> NPL[5 6]
[  5,  6] :    S(107)  -{ 3259}-> NPL[5 6]
[  6,  7] :    0(  0)  -{   -1}-> =E=

-------------------------------------


the bogey was destroyed by friendly aircraft

---------- chart ------------------
[  0,  1] :    0(  0)  -{   -1}-> =S=
[  1,  2] :   DT(  0)  -{   -1}-> the
[  1,  2] :  NPL( 49)  -{37495}-> DT[1 2]
[  1,  2] :   SS(103)  -{25363}-> NPL[1 2]
[  1,  2] :   NP( 90)  -{17837}-> NPL[1 2]
[  1,  2] :    S(135)  -{ 3259}-> NPL[1 2]
[  1,  3] :  NPL( 19)  -{38966}-> DT[1 2] NNX[2 3]
[  1,  3] :   NP( 60)  -{17837}-> NPL[1 3]
[  1,  3] :   SS( 73)  -{25363}-> NPL[1 3]
[  1,  3] :    S(105)  -{ 3259}-> NPL[1 3]
[  1,  4] :  NPL( 91)  -{39163}-> DT[1 2] NNX[2 3] VBX[3 4]
[  1,  4] :   SS( 53)  -{28334}-> NPL[1 3] VBX[3 4]
[  1,  4] :   NP(113)  -{21309}-> NPL[1 3] VBX[3 4]
[  1,  4] :    S( 83)  -{ 5358}-> NPL[1 3] VBX[3 4]
[  1,  5] :   SS( 71)  -{30731}-> NPL[1 3] VBX[3 4] VBN[4 5]
[  1,  5] :    S( 97)  -{ 7827}-> NPL[1 3] VBX[3 4] VBN[4 5]
[  1,  5] :   NP(141)  -{17854}-> NPL[1 3] SS[3 5]
[  1,  6] :   SS(112)  -{30939}-> NPL[1 3] VBX[3 4] VBN[4 5] IN[5 6]
[  1,  6] :    S(130)  -{ 8037}-> NPL[1 3] VBX[3 4] VBN[4 5] IN[5 6]
[  1,  6] :   NP(203)  -{17824}-> SS[1 6]
[  1,  7] :   SS(132)  -{31024}-> NPL[1 3] VBX[3 4] VBN[4 5] IN[5 6] JJ[6
7]
[  1,  8] :   SS(108)  -{30970}-> NPL[1 3] VBX[3 4] VBN[4 5] IN[5 6] NPL[6
8]
[  1,  8] :    S(120)  -{ 8081}-> NPL[1 3] VBX[3 4] VBN[4 5] IN[5 6] NPL[6
8]
[  1,  8] :   NP(199)  -{17824}-> SS[1 8]
[  2,  3] :  NNX(  0)  -{   -1}-> bogey
[  2,  3] :  NPL( 21)  -{41257}-> NNX[2 3]
[  2,  3] :   SS( 75)  -{25363}-> NPL[2 3]
[  2,  3] :   NP( 62)  -{17837}-> NPL[2 3]
[  2,  3] :    S(107)  -{ 3259}-> NPL[2 3]
[  2,  4] :  NPL(104)  -{41467}-> NNX[2 3] VBX[3 4]
```

```
[  2,  4] :   SS( 55)  -{28334}-> NPL[2 3] VBX[3 4]
[  2,  4] :   NP(115)  -{21309}-> NPL[2 3] VBX[3 4]
[  2,  4] :    S( 85)  -{ 5358}-> NPL[2 3] VBX[3 4]
[  2,  5] :   SS( 73)  -{30731}-> NPL[2 3] VBX[3 4] VBN[4 5]
[  2,  5] :    S( 99)  -{ 7827}-> NPL[2 3] VBX[3 4] VBN[4 5]
[  2,  5] :   NP(143)  -{17854}-> NPL[2 3] SS[3 5]
[  2,  6] :   SS(114)  -{30939}-> NPL[2 3] VBX[3 4] VBN[4 5] IN[5 6]
[  2,  6] :    S(132)  -{ 8037}-> NPL[2 3] VBX[3 4] VBN[4 5] IN[5 6]
[  2,  6] :   NP(205)  -{17824}-> SS[2 6]
[  2,  7] :   SS(134)  -{31024}-> NPL[2 3] VBX[3 4] VBN[4 5] IN[5 6] JJ[6
7]
[  2,  8] :   SS(110)  -{30970}-> NPL[2 3] VBX[3 4] VBN[4 5] IN[5 6] NPL[6
8]
[  2,  8] :    S(122)  -{ 8081}-> NPL[2 3] VBX[3 4] VBN[4 5] IN[5 6] NPL[6
8]
[  2,  8] :   NP(201)  -{17824}-> SS[2 8]
[  3,  4] :  VBX(  0)  -{   -1}-> was
[  3,  4] :  NPL( 92)  -{41473}-> VBX[3 4]
[  3,  4] :   SS( 70)  -{35435}-> VBX[3 4]
[  3,  4] :    S( 96)  -{15968}-> VBX[3 4]
[  3,  4] :   NP(133)  -{17837}-> NPL[3 4]
[  3,  5] :   SS( 78)  -{36227}-> VBX[3 4] VBN[4 5]
[  3,  5] :   NP(163)  -{20757}-> NPL[3 4] VBN[4 5]
[  3,  5] :    S(150)  -{15974}-> VBX[3 4] SS[4 5]
[  3,  8] :   SS( 96)  -{36306}-> VBX[3 4] VBN[4 5] IN[5 6] NPL[6 8]
[  3,  8] :    S(126)  -{16090}-> VBX[3 4] VBN[4 5] IN[5 6] NPL[6 8]
[  3,  8] :   NP(187)  -{17824}-> SS[3 8]
[  4,  5] :  VBX( 10)  -{   -1}-> destroyed
[  4,  5] :  VBN(  4)  -{   -1}-> destroyed
[  4,  5] :  NPL(102)  -{41473}-> VBX[4 5]
[  4,  5] :   SS( 67)  -{35156}-> VBN[4 5]
[  4,  5] :    S( 97)  -{15622}-> VBN[4 5]
[  4,  5] :   NP(143)  -{17837}-> NPL[4 5]
[  4,  6] :   SS(113)  -{35184}-> VBN[4 5] IN[5 6]
[  4,  6] :   NP(204)  -{17824}-> SS[4 6]
[  4,  6] :    S(209)  -{ 2663}-> SS[4 6]
[  4,  7] :   SS(113)  -{35222}-> VBN[4 5] IN[5 6] JJ[6 7]
[  4,  7] :   NP(204)  -{17824}-> SS[4 7]
[  4,  7] :    S(209)  -{ 2663}-> SS[4 7]
[  4,  8] :   SS( 95)  -{35198}-> VBN[4 5] IN[5 6] NPL[6 8]
[  4,  8] :    S(114)  -{15649}-> VBN[4 5] IN[5 6] NPL[6 8]
[  4,  8] :   NP(186)  -{17824}-> SS[4 8]
[  5,  6] :   IN(  0)  -{   -1}-> by
[  5,  6] :  NPL(111)  -{39253}-> IN[5 6]
[  5,  6] :   SS(109)  -{32171}-> IN[5 6]
[  5,  6] :   NP(152)  -{17837}-> NPL[5 6]
[  5,  6] :    S(197)  -{ 3259}-> NPL[5 6]
[  5,  7] :  NPL(125)  -{39284}-> IN[5 6] JJ[6 7]
[  5,  7] :   SS(146)  -{32403}-> IN[5 6] NPL[6 7]
[  5,  7] :   NP(165)  -{22524}-> IN[5 6] NPL[6 7]
[  5,  8] :   SS(114)  -{32403}-> IN[5 6] NPL[6 8]
[  5,  8] :   NP(133)  -{22524}-> IN[5 6] NPL[6 8]
[  5,  8] :  NPL(105)  -{39286}-> IN[5 6] JJ[6 7] NNX[7 8]
[  5,  8] :    S(191)  -{ 3259}-> NPL[5 8]
[  6,  7] :   JJ(  0)  -{   -1}-> friendly
[  6,  7] :  NPL( 61)  -{39361}-> JJ[6 7]
[  6,  7] :   SS( 57)  -{32827}-> JJ[6 7]
[  6,  7] :    S(107)  -{14007}-> JJ[6 7]
[  6,  7] :   NP(102)  -{17837}-> NPL[6 7]
```

```
[  6,  8] :  NPL( 29)  -{39657}-> JJ[6 7] NNX[7 8]
[  6,  8] :   SS( 83)  -{25363}-> NPL[6 8]
[  6,  8] :   NP( 70)  -{17837}-> NPL[6 8]
[  6,  8] :    S(115)  -{ 3259}-> NPL[6 8]
[  7,  8] :  NNX(  0)  -{   -1}-> aircraft
[  7,  8] :  NPL( 21)  -{41257}-> NNX[7 8]
[  7,  8] :   SS( 75)  -{25363}-> NPL[7 8]
[  7,  8] :   NP( 62)  -{17837}-> NPL[7 8]
[  7,  8] :    S(107)  -{ 3259}-> NPL[7 8]
[  8,  9] :    0(  0)  -{   -1}-> =E=
------------------------------------
```

The following is only a partial chart parse results. It only shows the nodes extending from node 1.

military deception can be used to enhance the impact obtained from practicing these tenets, but military deception also provides opportunities to apply these tenets in ways that are not otherwise available

```
---------- chart ------------------
[  1,  2] :  NNX( 21)  -{   -1}-> military
[  1,  2] :   JJ(  1)  -{   -1}-> military
[  1,  2] :  NPL( 42)  -{41257}-> NNX[1 2]
[  1,  2] :   SS( 58)  -{32827}-> JJ[1 2]
[  1,  2] :    S(108)  -{14007}-> JJ[1 2]
[  1,  2] :   NP( 83)  -{17837}-> NPL[1 2]
[  1,  3] :  NPL( 30)  -{39657}-> JJ[1 2] NNX[2 3]
[  1,  3] :   SS( 84)  -{25363}-> NPL[1 3]
[  1,  3] :   NP( 71)  -{17837}-> NPL[1 3]
[  1,  3] :    S(116)  -{ 3259}-> NPL[1 3]
[  1,  4] :   SS(101)  -{25898}-> NPL[1 3] MD[3 4]
[  1,  4] :    S(137)  -{ 4126}-> NPL[1 3] MD[3 4]
[  1,  4] :   NP(178)  -{17854}-> NPL[1 2] SS[2 4]
[  1,  5] :   SS( 84)  -{26173}-> NPL[1 3] MD[3 4] VB[4 5]
[  1,  5] :    S(112)  -{ 4263}-> NPL[1 3] MD[3 4] VB[4 5]
[  1,  5] :   NP(159)  -{17854}-> NPL[1 3] SS[3 5]
[  1,  6] :   SS( 95)  -{26826}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
[  1,  6] :    S(125)  -{ 4572}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
[  1,  6] :   NP(167)  -{17854}-> NPL[1 3] SS[3 6]
[  1,  7] :   SS(172)  -{26857}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 7]
[  1,  7] :    S(182)  -{ 4587}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 7]
[  1,  7] :   NP(227)  -{17854}-> NPL[1 3] SS[3 7]
[  1,  8] :   SS(134)  -{26857}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 8]
[  1,  8] :    S(144)  -{ 4587}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 8]
[  1,  8] :   NP(189)  -{17854}-> NPL[1 3] SS[3 8]
[  1,  9] :   SS(171)  -{26857}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 9]
[  1,  9] :    S(181)  -{ 4587}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 9]
[  1,  9] :   NP(226)  -{17854}-> NPL[1 3] SS[3 9]
[  1, 10] :   SS(141)  -{26857}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 10]
[  1, 10] :    S(151)  -{ 4587}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 10]
```

```
[  1, 10] :   NP(196)  -{17854}-> NPL[1 3] SS[3 10]
[  1, 11] :    SS(200)  -{26857}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 11]
[  1, 11] :     S(210)  -{ 4587}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 11]
[  1, 11] :   NP(255)  -{17854}-> NPL[1 3] SS[3 11]
[  1, 12] :    SS(246)  -{26857}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 12]
[  1, 12] :     S(256)  -{ 4587}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 12]
[  1, 12] :   NP(301)  -{17854}-> NPL[1 3] SS[3 12]
[  1, 13] :    SS(235)  -{26857}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 13]
[  1, 13] :     S(245)  -{ 4587}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 13]
[  1, 13] :   NP(290)  -{17854}-> NPL[1 3] SS[3 13]
[  1, 14] :    SS(309)  -{26857}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 14]
[  1, 14] :     S(319)  -{ 4587}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 14]
[  1, 14] :   NP(364)  -{17854}-> NPL[1 3] SS[3 14]
[  1, 15] :    SS(279)  -{26857}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 15]
[  1, 15] :     S(289)  -{ 4587}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 15]
[  1, 15] :   NP(334)  -{17854}-> NPL[1 3] SS[3 15]
[  1, 16] :    SS(356)  -{26857}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 16]
[  1, 16] :     S(366)  -{ 4587}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 16]
[  1, 16] :   NP(411)  -{17854}-> NPL[1 3] SS[3 16]
[  1, 17] :    SS(447)  -{26857}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 17]
[  1, 17] :     S(457)  -{ 4587}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 17]
[  1, 17] :   NP(502)  -{17854}-> NPL[1 3] SS[3 17]
[  1, 18] :    SS(389)  -{26857}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 18]
[  1, 18] :    S(372)  -{ 3010}-> SS[1 15] ,[15 16] CC[16 17] SS[17 18]
[  1, 18] :   NP(444)  -{17854}-> NPL[1 3] SS[3 18]
[  1, 19] :    SS(381)  -{26857}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 19]
[  1, 19] :     S(391)  -{ 4587}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 19]
[  1, 19] :   NP(436)  -{17854}-> NPL[1 3] SS[3 19]
[  1, 20] :   NP(476)  -{17854}-> NPL[1 3] SS[3 20]
[  1, 20] :    SS(421)  -{26857}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 20]
[  1, 20] :    S(425)  -{ 3010}-> SS[1 15] ,[15 16] CC[16 17] SS[17 20]
[  1, 21] :    SS(426)  -{26857}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 21]
[  1, 21] :    S(409)  -{ 3010}-> SS[1 15] ,[15 16] CC[16 17] SS[17 21]
[  1, 21] :   NP(481)  -{17854}-> NPL[1 3] SS[3 21]
[  1, 22] :    SS(448)  -{26857}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 22]
[  1, 22] :    S(431)  -{ 3010}-> SS[1 15] ,[15 16] CC[16 17] SS[17 22]
[  1, 22] :   NP(503)  -{17854}-> NPL[1 3] SS[3 22]
[  1, 23] :   NP(598)  -{17854}-> NPL[1 3] SS[3 23]
[  1, 23] :    SS(543)  -{26857}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
```

```
TOINF[6 23]
[  1, 23] :    S(529)  -{ 3010}-> SS[1 15] ,[15 16] CC[16 17] SS[17 23]
[  1, 24] :   NP(561)  -{17854}-> NPL[1 3] SS[3 24]
[  1, 24] :    SS(506)  -{26857}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 24]
[  1, 24] :    S(492)  -{ 3010}-> SS[1 15] ,[15 16] CC[16 17] SS[17 24]
[  1, 25] :   NP(598)  -{17854}-> NPL[1 3] SS[3 25]
[  1, 25] :    SS(543)  -{26857}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 25]
[  1, 25] :    S(529)  -{ 3010}-> SS[1 15] ,[15 16] CC[16 17] SS[17 25]
[  1, 26] :   NP(568)  -{17854}-> NPL[1 3] SS[3 26]
[  1, 26] :    SS(513)  -{26857}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 26]
[  1, 26] :    S(499)  -{ 3010}-> SS[1 15] ,[15 16] CC[16 17] SS[17 26]
[  1, 27] :   NP(630)  -{17854}-> NPL[1 3] SS[3 27]
[  1, 27] :    SS(575)  -{26857}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 27]
[  1, 27] :    S(560)  -{ 3010}-> SS[1 15] ,[15 16] CC[16 17] SS[17 27]
[  1, 28] :   NP(604)  -{17854}-> NPL[1 3] SS[3 28]
[  1, 28] :    SS(549)  -{26857}-> NPL[1 3] MD[3 4] VB[4 5] VBN[5 6]
TOINF[6 28]
[  1, 28] :    S(535)  -{ 3010}-> SS[1 15] ,[15 16] CC[16 17] SS[17 28]
[  1, 29] :   SS(649)  -{25218}-> SS[1 15] ,[15 16] CC[16 17] SS[17 29]
[  1, 29] :    S(632)  -{ 3010}-> SS[1 15] ,[15 16] CC[16 17] SS[17 29]
[  1, 29] :   NP(716)  -{17827}-> SS[1 15] ,[15 16] CC[16 17] NP[17 29]
```

Appendix D

D Apple Pie Parser Nicknames

1 S
2 ADJP
3 ADJP-ADV
4 ADJP-CLR
5 ADJP-HLN
6 ADJP-LOC
7 ADJP-MNR
8 ADJP-PRD
9 ADJP-PRD-TPC
10 ADJP-SBJ
11 ADJP-TMP
12 ADJP-TPC
13 ADJP-TPC-PRD
14 ADJP-TTL
15 ADVP
16 ADVP-CLR
17 ADVP-CLR-MNR
18 ADVP-CLR-TPC
19 ADVP-DIR
20 ADVP-DIR-CLR
21 ADVP-DIR-TPC
22 ADVP-EXT
23 ADVP-HLN
24 ADVP-LOC
25 ADVP-LOC-CLR
26 ADVP-LOC-CLR-TPC
27 ADVP-LOC-PRD
28 ADVP-LOC-PRD-TPC
29 ADVP-LOC-TMP
30 ADVP-LOC-TPC
31 ADVP-LOC-TPC-PRD
32 ADVP-MNR
33 ADVP-MNR-CLR
34 ADVP-MNR-TMP
35 ADVP-MNR-TPC
36 ADVP-PRD
37 ADVP-PRD-LOC
38 ADVP-PRD-LOC-TPC
39 ADVP-PRD-TMP
40 ADVP-PRD-TPC
41 ADVP-PRP
42 ADVP-PUT
43 ADVP-PUT-TPC
44 ADVP-TMP
45 ADVP-TMP-CLR
46 ADVP-TMP-CLR-TPC
47 ADVP-TMP-PRD
48 ADVP-TMP-TPC
49 ADVP-TPC
50 ADVP-TPC-PRD
51 ADVP|PRT
52 CONJP
53 FRAG
54 FRAG-ADV
55 FRAG-HLN
56 FRAG-PRD
57 FRAG-TPC
58 FRAG-TTL
59 FRAG-TTL-SBJ
60 INTJ
61 INTJ-CLR
62 INTJ-HLN
63 LST
64 NAC
65 NAC-LOC
66 NAC-TMP
67 NAC-TTL
68 NP
69 NP-ADV
70 NP-BNF
71 NP-CLR
72 NP-CLR-LOC
73 NP-CLR-TMP
74 NP-DIR
75 NP-EXT
76 NP-HLN
77 NP-LGS
78 NP-LOC
79 NP-LOC-CLR
80 NP-LOC-HLN
81 NP-LOC-PRD
82 NP-LOC-PRD-TPC
83 NP-LOC-TPC-PRD
84 NP-MNR
85 NP-MNR-CLR
86 NP-PRD
87 NP-PRD-TPC
88 NP-PRD-TTL
89 NP-SBJ
90 NP-SBJ-TTL
91 NP-TMP
92 NP-TMP-CLR
93 NP-TMP-HLN
94 NP-TMP-PRD
95 NP-TMP-TPC
96 NP-TPC
97 NP-TTL
98 NP-TTL-PRD
99 NP-TTL-SBJ
100 NP-TTL-TPC
101 NP-VOC
102 NX
103 NX-TTL
104 PP
105 PP-BNF
106 PP-CLR
107 PP-CLR-LOC
108 PP-CLR-TMP
109 PP-CLR-TPC
110 PP-DIR
111 PP-DIR-CLR
112 PP-DIR-PRD
113 PP-DTV
114 PP-EXT
115 PP-HLN
116 PP-LGS
117 PP-LOC
118 PP-LOC-CLR
119 PP-LOC-CLR-TPC
120 PP-LOC-HLN
121 PP-LOC-MNR
122 PP-LOC-PRD
123 PP-LOC-PRD-TPC
124 PP-LOC-TPC
125 PP-LOC-TPC-PRD
126 PP-MNR
127 PP-MNR-CLR
128 PP-MNR-PRD
129 PP-NOM
130 PP-PRD
131 PP-PRD-LOC
132 PP-PRD-LOC-TPC
133 PP-PRD-TPC
134 PP-PRP
135 PP-PRP-CLR
136 PP-PRP-PRD
137 PP-PUT
138 PP-SBJ
139 PP-TMP
140 PP-TMP-CLR
141 PP-TMP-PRD
142 PP-TMP-TPC
143 PP-TPC
144 PP-TPC-CLR
145 PP-TPC-LOC-PRD
146 PP-TPC-PRD
147 PP-TTL
148 PP-TTL-PRD
149 PRN
150 PRT
151 PRT|ADVP
152 QP
153 RRC
154 S-ADV
155 S-CLF
156 S-CLF-TPC
157 S-CLR
158 S-CLR-ADV
159 S-HLN
160 S-LOC
161 S-MNR
162 S-MNR-CLR
163 S-NOM
164 S-NOM-LGS
165 S-NOM-PRD
166 S-NOM-SBJ
167 S-PRD
168 S-PRD-TPC
169 S-PRP
170 S-PRP-CLR
171 S-PRP-PRD
172 S-PRP-TPC
173 S-SBJ
174 S-TMP
175 S-TMP-TPC
176 S-TPC
177 S-TPC-TMP
178 S-TTL
179 S-TTL-PRD
180 S-TTL-SBJ
181 SBAR
182 SBAR-ADV
183 SBAR-ADV-TPC
184 SBAR-CLR
185 SBAR-DIR
186 SBAR-DIR-TPC
187 SBAR-HLN
188 SBAR-LOC
189 SBAR-LOC-CLR
190 SBAR-LOC-PRD
191 SBAR-MNR
192 SBAR-MNR-PRD
193 SBAR-NOM
194 SBAR-NOM-LGS
195 SBAR-NOM-PRD
196 SBAR-NOM-SBJ

197 SBAR-NOM-TPC
198 SBAR-PRD
199 SBAR-PRD-TPC
200 SBAR-PRP
201 SBAR-PRP-PRD
202 SBAR-PUT
203 SBAR-SBJ
204 SBAR-TMP
205 SBAR-TMP-CLR
206 SBAR-TMP-PRD
207 SBAR-TPC
208 SBAR-TTL
209 SBARQ
210 SBARQ-HLN
211 SBARQ-NOM
212 SBARQ-PRD
213 SBARQ-TPC
214 SBARQ-TTL
215 SINV
216 SINV-ADV
217 SINV-HLN
218 SINV-TPC
219 SINV-TTL
220 SQ
221 SQ-PRD
222 SQ-TPC
223 SQ-TTL
224 UCP
225 UCP-ADV
226 UCP-CLR
227 UCP-DIR
228 UCP-EXT
229 UCP-HLN
230 UCP-LOC
231 UCP-LOC-CLR
232 UCP-LOC-PRD
233 UCP-MNR
234 UCP-PRD
235 UCP-PRD-LOC
236 UCP-PRP
237 UCP-TMP
238 UCP-TPC
239 VP
240 VP-TPC
241 VP-TTL
242 WHADJP
243 WHADVP
244 WHADVP-TMP
245 WHNP
246 WHPP
247 X
248 X-ADV
249 X-CLF
250 X-DIR
251 X-EXT
252 X-HLN
253 X-PUT
254 X-TMP
255 X-TTL
261 AUX
262 NEG
271 SS
272 NPL
281 TOINF
401 #
402 $
403 ``
404 -LRB-
405 -RRB-
406 ,
407 .
408 :
409 CC
410 CD
411 DT
412 EX
413 FW
414 IN
415 JJ
416 JJR
417 JJS
418 LS
419 MD
420 NN
421 NNP
422 NNPS
423 NNS
424 PDT
425 POS
426 PRP
427 PRP$
428 RB
429 RBR
430 RBS
431 RP
432 SYM
433 TO
434 UH
435 VB
436 VBD
437 VBG
438 VBN
439 VBP
440 VBZ
441 WDT
442 WP
443 WP$
444 WRB
445 ''
446 -NONE-
481 NNPX
482 NNX
483 VBX
521 *NNP
522 *CD
523 *FW
524 *SYM
561 @OF
562 @SNC
563 @DLQ
564 @HAVE
565 @BE
566 @DLQ2
771 (
772)
773 -S-
774 -E-

Bibliography

[1] Allen, James. *Natural Language Understanding*. The Bejamin/Cummings Publishing Company, Inc, Redwood City, CA 1995.

[2] Fromkin, Victoria and Rodman, Robert. *An Introduction to Language*. Ho' ., Rinehart and Winston, Inc, Orlando, FL 32887.

[3] Hutchins, John W. and Somers, Harold L. *An Introduction to Machine Translation.* Academic Press Limited, San Diego, CA 1992.

[4] Lee, Young-Suk, Weinstein, Clifford, Seneff, Stephanie, and Tummala, Dinesh. An Application of Part-of-Speech Tagging to Robust Parsing in Machine Translation of Telegraphic Messages Unpublished Manuscript. MIT Lincoln Laboratories, 1996.

[5] Lee, Young-Suk. Automated English/Korean Translation for Enhanced Coalition Communications. JAC Technical Seminar, MIT Lincoln Laboratories, March 1997.

[6] Lee, Young-Suk. Project Report 2/25/97. MIT Lincoln Laboratories, 1997

[7] Lee, Young-Suk, Weinstein, Clifford, Seneff, Stephanie, and Tummala, Dinesh. Ambiguity Resolution for Machine Translation. Proceedings of the 35th Conference of the Association of Computational Linguistics (ACL '97), 1997.

[8] Sekine, Satoshi and Grishman, Ralph. A Corpus-based Probabilistic Grammar with Only Two Non-terminals. Fourth International Workshop on Parsing Technology,

1995.

[9] Sekine, Satoshi. *Manual of Apple Pie Parser*. New York University, NY 1996

[10] Seneff, Stephanie. TINA: A Natural Language System for Spoken Language Applications. Association for Computational Linguistics, 1992.

[11] Tummala, Dinesh, Seneff, Stephanie, Paul, Douglas, Weinstein, Clifford, and Yang, Dennis. CCLINC: System Architecture and Concept Demonstration of Speech-to-Speech Translation for Limited-Domain Multilingual Applications. ARPA Spoken Language Technology Workshop, 1995.

[12] Weinstein, Clifford, Tummala, Dinesh, Lee, Young-Suk, and Seneff, Stephanie. Automatic English-to-Korean Text Translation of Telegraphics Messages in a Limited Domain. COLING '96. Copenhagen, Denmark 1996.

[13] Weinstein, Clifford, Lee, Young-Suk, Seneff, Stephanie, and Tummala, Dinesh. Presentation at CSTAR II Meeting: Automatic English/Korea Translation of Telegraphics Messages in a Limited Domain. Kyoto, Japan 1996.