Application of Part-of-Speech Tagger in Robust Machine Translation System

by

Sung S. Park

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Electrical Science and Engineering

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 22, 1998

Author ....
Electrical Engineering and Computer Science
May 18, 1998

Certified by ....
Thesis Supervisor: Dr. Clifford J. Weinstein
Title: Group Leader, Information Systems Technology, Lincoln Laboratory

Certified by ....
Thesis Supervisor: Dr. Young-Suk Lee
Title: Staff Member, Information Systems Technology, Lincoln Laboratory

Accepted by ....
Arthur C. Smith
Chairman, Deprtment Committee on Graduate Theses

Application of Part-of-Speech Tagger in Robust Machine Translation System
by

Sung S. Park

Submitted to the
Department of Electrical Engineering and Computer Science

May 22, 1998

in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Electrical Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

In an attempt to obtain robust machine translation system, a part-of-speech tagger system was explored. A rule based tagger system was selected to be used and the performance of the system in the MUCII domain was evaluated. An abstract representation of the tagger system was presented to examine reusability of the system in enhancing the machine translation system. The part-of-speech tagger was integrated into the machine translation system for effective operation on a sentence-by-sentence basis.

Thesis Supervisor: Dr. Cliff J. Weinstein
Title: Group Leader, Information Systems Technology, Lincoln Laboratory

Thesis Supervisor: Dr. Young-Suk Lee
Title: Staff Member, Information Systems Technology, Lincoln Laboratory

Contents

List of Figures

List of Tables

Chapter 1
Overview

1.1 Introduction

For telegraphic military message communication, Information Systems Technology Group at Lincoln Laboratory has developed an interlingua based English to Korean machine translation system[1][2][3]. The system utilizes a meaning representation in the form of semantic frame as the interlingua. It carries out language translation in two steps; analysis of source language to produce a semantic frame, and generation of target language from the semantic frame. TINA[6] and GENESIS[7][2] are the analysis and generation subsystems. Both TINA and GENESIS were developed at Laboratory for Computer Science, MIT.

Early versions of the translation system achieved high quality translation by limiting the domain to be translated and lexicalizing grammar rules defined in terms of semantic categories. However, highly lexicalized grammars have relatively low coverage, and resulted in low parsing coverage over all. This caused the system to fail upon encountering sentences with words or constructions unknown to the system.

An obvious solution to the unknown word problem is to extend the lexical and syntactic knowledge of the system by increasing the size of the lexicon. However, this solution is problematic in that most unknown word are open class items such as nouns, verbs, adjectives and adverbs, which is not limited in number. Therefore, it is almost impossible to come up with a lexicon which covers every possible word.

Another solution which has been pursued by the Group at Lincoln Laboratory, and which is the topic of this thesis, is to utilize an automatic part-of-speech tagger. By using such tagger, the system can inexpensively identify the parts-of-speech of unknown words or words in unknown construction, and by doing so the system can have an improved parsing coverage. In this approach, the system replaces each unknown word by its part-of-speech, instead of the actual words, to the analysis grammar. The replaced parts-of-speech are handled by a combined syntactic/ semantic grammar augmented for generic open class items. The replaced parts-of-speech are handled by semantic grammar augmented for generic open class items. The technique is to include just enough semantic information to solve ambiguities, by anchoring words that have high semantic relevance within the domain. However, the current grammar is left intact since it efficiently parses even highly telegraphic messages. For more detailed information on this method, refer to reports [4] and [5].

The approach of utilizing a tagger system to enhance the machine translation system poses two questions. One is in the selection of a tagger system. Since the performance of the enhanced machine translation system for sentences with words and constructions unknown to the system heavily depends on the accuracy of the part-of-speech tagger, a tagger system which is adaptable to the translation domain is crucial in achieving high quality parsing. A desired tagger system should be trainable to extract rules specific in the domain, and should achieve high accuracy from the learned rules.

A rule based tagger developed by Brill[8][9] was evaluated to get an assessment of the usability in the domain of telegraphic messages. The system was initially trained and evaluated, and showed good performances. The tagger uses both lexical rules and contextual rules to find the appropriate parts-of-speech of unknown words. The system is trained through transformation-based error-driven learning. More about the transformation-based error-driven learning is discussed in Chapter 2. The system extracts critical linguistic information and captures the information in a set of rules. The learned rules are small in numbers. They are simple, yet powerfully effective.

Given a tagger system, the next question is how to incorporate the tagger system to enhance the machine translation system. The incorporation involves multiple steps. First, the usability of the tagger system for machine translation system should be investigated. The overall usability of the system in the domain was tested, and the system tagged each word in high accuracy. However, in order to incorporate the system into machine translation system, a thorough analysis of each functionality of the tagger system is necessary. Through the analysis, complete abstract representations of each functionality can be constructed, and usability of each functionality can be identified in the context of the intended application. The functionalities identified needing changes for use in the machine translation application should be modified to obtain an overall system that behaves within the desired specification. These steps are captured in the software reengineering approach[11]. The approach consists three parts;

reverse engineering, *change*, and *forward engineering.* The *reverse engineering,* also called understanding is constructing abstract representations of the system being reengineered. *Change* is to add new functionality or to modify existing functionality to adapt to new application or to modify the system architecture. *Forward engineering* is taking requirements of completing the system and obtaining an executable system. The current thesis is focused in the first and second steps of reengineering of the rule based tagger system.

The analysis of the tagger system revealed a problem in integration of the system interface with the machine translation system. The tagger system is comprised of two subsystems interfaced with an unidirectional stream. The subsystem interface is organized such that the whole tagger system starts running when given a file of input sentences and completes after processing the file. When the system starts, the system initializes tables of fairly large sizes. The target machine translation system, however, requires an interface that enables the system to process one sentence after another efficiently. Since the ultimate target application is speech to speech translation[1], the waiting time for the tagger system to initialize its tables for every sentence is too costly. A system interface better suited for the target application is bidirectional streams between the machine translation system and the tagger system. In this way, the tagger system can be initialized once at the start of the machine translation system, and can return a string of a sentence tagged with parts-of-speech to the machine translation system whenever the machine translation system fails to parse due to unknown words or construction. As in a client/server system, the tagger system provides tagged sentences as a service to the request the machine translation system makes when the machine translation system fails to parse.

The integration of two systems into client/server system by modifying the system interface is discussed in detail in chapter three. In chapter two, the analysis of the tagger system is presented, and a complete abstract representation of each functionality of the tagger system is described.

In the following sections of this chapter, a brief description of each subsystem of the machine translation system and the rule-based tagger system is presented. The thesis concludes with a discussion of potential future directions for the research.

1.2 Subsystems

1.2.1 TINA

TINA is the subsystem for language understanding. The system is based on context-free grammars, augmented transition networks, and the unification concept. The system utilizes a context free grammar augmented with a set of features used to enforce syntactic and semantic constraints, providing a seamless interface between syntax and semantics through grammar rules which incorporate syntactic and semantic categories. Augmented transition networks are converted from the rules using a straightforward process. The unifications are performed in a one-dimensional framework[6].

Previous versions of the analysis portion of the translation system utilized highly lexicalized grammar rules to resolve the lexical ambiguities, which are inherent in highly elliptical sentences of telegraphic messages. However, highly lexicalized grammar rules are time consuming to construct, and have low parsing coverage except in narrow domains. In order to achieve higher grammar coverage, the system has been enhanced with grammars which deploys lexicalized semantic rules for ambiguity resolution and resorts to more general rules for unambiguous cases. If an input sentence fails to parse on the basis of the lexicalized semantic grammar rules due to unknown words or constructions, parsing of the part-of-speech sequence corresponding to the input word sequence will be tried[4][5].

1.2.2 GENESIS

The GENESIS system is a table-driven language generation system with mechanisms handling inflectional endings. The system is composed of three modules; lexicon, message, and rewrite rules[2][7].

Lexicon provides the surface form of a semantic frame entry including the inflectional ending. Each entry in the table is followed by the part of speech, stem of the word and derived forms. If necessary, grammatical attributes such as articles or auxiliary verb can be specified.

Rewrite rules captures the surface phonological constraints and contractions of the target languages.

Message templates are primarily used to recursively construct phrases describing the each nodes in semantic frame. A message template consists message name and sequence of one or more strings of key words.

1.2.3 TAGGER

For years, Markov-model based stochastic taggers have received more attentions from the researchers in automatic part of speech tagging, due to a number of advantages over the manually built taggers. As large corpora became available, stochastic taggers captured useful linguistic information, even indirectly in large tables of statistics, which human engineers fail to notice. These taggers also eliminated the need for laborious manual rule construction. However, Brill[5] described a trainable rule-based tagger that achieves comparable performance to that of stochastic taggers, and captures linguistic information directly in a set of simple non-stochastic rules. The tagger is based on a transformation-based error-driven learning enhanced with lexicalization and unknown word tagging.

The mechanism of the transformation-based error-driven part-of-speech tagger will be discussed in detail in Chapter 2.

The original tagger system is optimized for a corpus of moderate size. In consequence, the tagger is very inefficient for processing a sentence, which our application requires. So, The tagger system has been modified for enhanced interface with the translation system. The new tagger system is comprised of two processes with an Unix domain socket residing in a local file system, /tmp, as an interprocess communication pipe. The two processes resemble their respective predecessor in functionality, but with capability to process one sentence at a time. In chapter 3, the design and implementation of the modified tagger, which takes in input sentence from stdin and prints out output tagged sentence to the stdout, is illustrated in great detail.

1.3 Incorporation

To improve the performance of the Machine Translation system, the tagger system is incorporated into the existing translation system. The tagger system is reused to serve as tagger module in the translation system. The reuse of the software was achieved by software reengineering approach, where the system was analyzed for reusability, made modifications for adoption, and integrated into the translation system. The translation system begins by making connection to the tagger system, which is implemented as a server system utilizing UNIX Domain socket. The implementation of tagger system with inter-process interface is discussed in detail in Chapter 3.

The translation system consults the tagger system upon failure of parsing. The tagger system returns to the translation system the sentence tagged with its most likely part of speech. The translation system checks each word against the table of frequently occurring words, and annotate the word with an asterisk(*), and puts all words in a format of word*pos.

For example, the following string is prepared for reentry for parsing from the unparsed input sentence. Sterett, a name of a ship, is unknown word, and assume fire is present in the frequently occurring words table.

- input sentence:Sterett taken under fire
- retry stringSterett*NN taken*VBN under*IN *fire*NN.

1.4 Evaluation

At the end of each following chapter, an evaluation of the system is described. A rule-based tagger system is evaluated both before and after training on the MUC-II database.

Chapter 2 Part-Of-Speech Tagger

2.1 Introduction

The tagger described in [8] was evaluated as being well-suited for application in our translation domain. The benchmark result of the performance with respect to the MUCII Data[1] displayed high rates of tagging accuracy. The adoption of part-of-speech tagging technique has proven to be effective in enhancing the parsing coverage of the translation system, and the incorporation of the tagger into the existing translation system has been further studied.

In the following sections of this chapter, the mechanisms of the transformation-based part-of-speech tagger are explored in detail to help readers better understand what procedures were taken, and what was changed, in order to incorporate the tagger system with the Machine Translation System. Section 2.2 describes the tagger in detail, and in section 2.3, the training of the tagger system in MUCII Data is illustrated in detail.

2.2 Transformation-based part-of-speech tagger

1 MUC-II stands for the Second Message Understanding Conference. MUC-II messages were originally collected and prepared by NRaD(1989) to support DARPA-sponsored research in message understanding

The tagger system is comprised of two subsystems; the start-state-tagger and the final-state-tagger. Each system can run independently, but when the two systems are run together, the output of the start-state-tagger is pipelined to the input of the final-state-tagger. Such interface between the two subsystems is not suitable to be incorporated into the Machine Translation System. Chapter 3 describes the modification to the interface. However, before making modifications, the functionality of the tagger system should be analyzed in detail. This section provides comprehensive analysis of the start-state-tagger and the final-state-tagger.

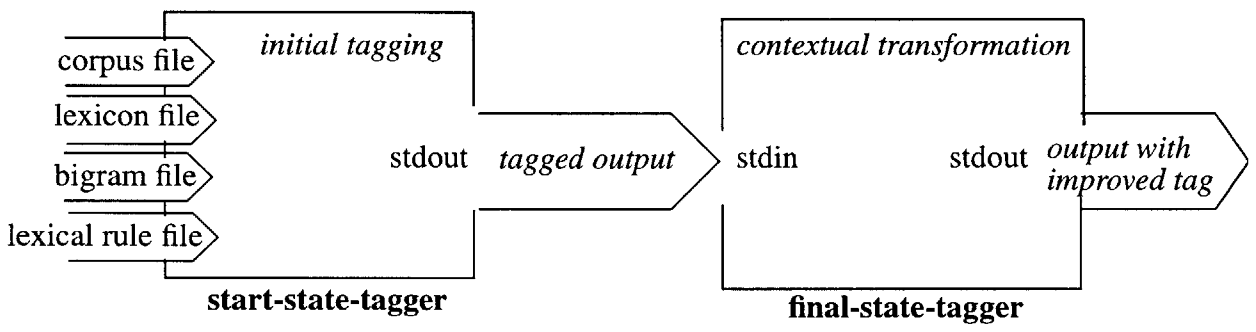


FIGURE 2-1 Data Flow of Tagger

The brief overview of the system is as follows. The start-state-tagger assigns every word its most likely tag in isolation. Each word, if it is a member of a lexicon provided to the system, has a lexical entry consisting of a partially ordered list of tags, indicating the most likely tag for that word, as well as all other tags seen with that word in no particular order. The start-state-tagger only looks for the most likely tag. Other tags are used by the final-state-tagger. Since it matters only that a tag is seen with a word, the final-state-tagger does not care for the frequency that the tag appears with the word, and the tags do not need to be ordered.

For unknown words, i.e. words not in the lexicon, a list of transformations is provided to the start-state-tagger to determine the most likely tag. Unknown words are first assumed to be nouns, and then, cues based upon prefixes, suffixes, infixes, and adjacent word co-occurrence are used to modify the initial assumption

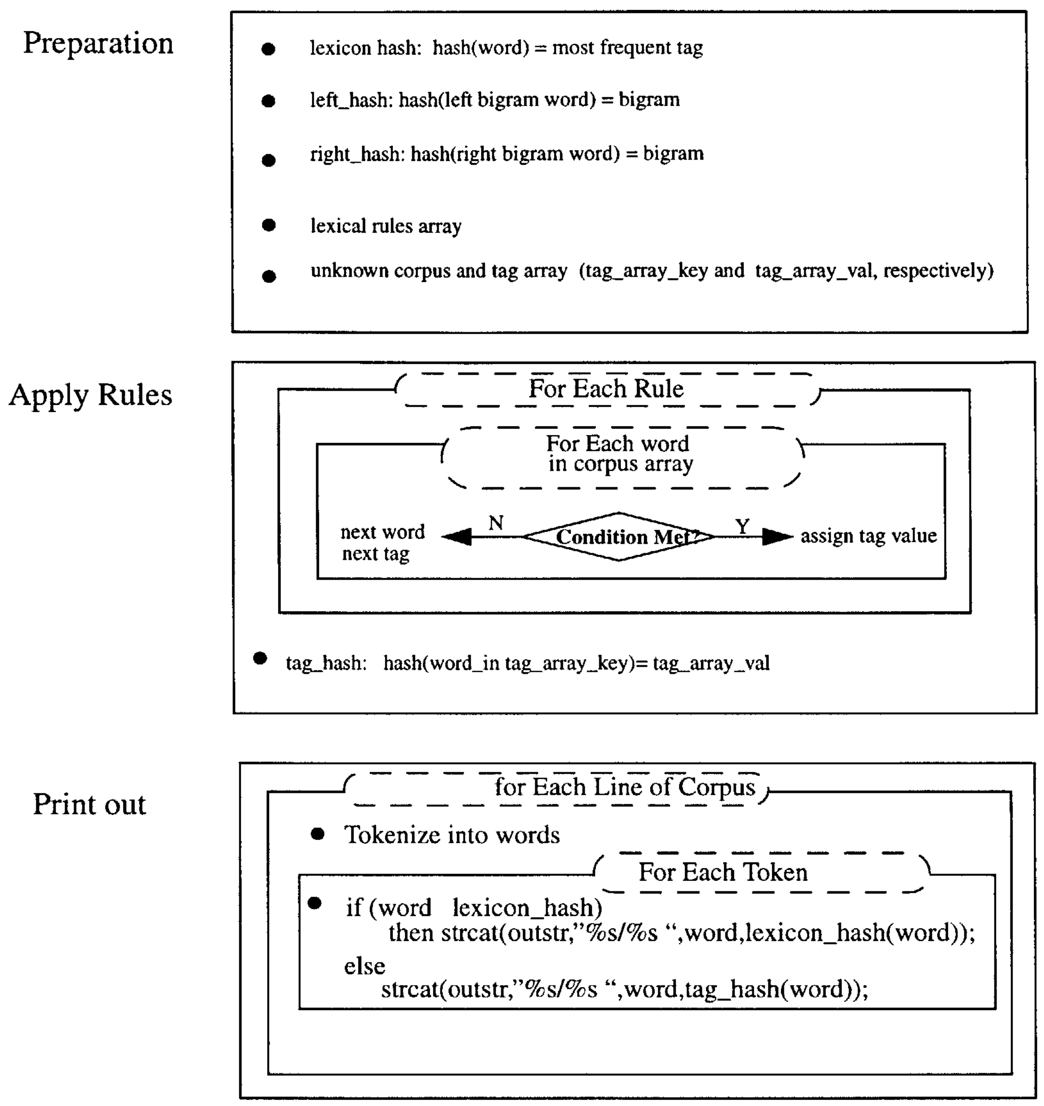
The final-state-tagger uses contextual transformations to improve accuracy of the tagged corpus.

The following sections describe the detailed functionality of the two sub-systems.

2.2.1 The start-state-tagger

The start-state-tagger needs four files as arguments: LEXICON, CORPUS, BIGRAM, and LEXICALRULES. LEXICON consists of words and a list of tags for each word. The most frequent tag for the corresponding word comes first, and the rest in no particular order. CORPUS contains the text of sentences to be tagged. BIGRAM has a list of frequently occurring adjacent words. Finally, LEXICALRULES contains cues such as prefixes, suffixes, infixes, and adjacent word co-occurrences, to find the most likely tags of unknown words.

The start-state-tagger operates in three stages: preparation, rule application, and print out. Figure 2-2. describes the stages and the functionalities.


Preparation
lexicon hash: hash(word) = most frequent tag
left_hash: hash(left bigram word) = bigram
right_hash: hash(right bigram word) = bigram
lexical rules array
unknown corpus and tag array (tag_array_key and tag_array_val, respectively)
Apply Rules
For Each Rule
For Each word in corpus array
next word next tag
N
Condition Met?
Y
assign tag value
tag_hash: hash(word_in tag_array_key)= tag_array_val
Print out
for Each Line of Corpus
Tokenize into words
For Each Token

```
if (word   lexicon_hash)
     then strcat(outstr,"%s/%s ",word,lexicon_hash(word));
else
    strcat(outstr,"%s/%s ",word,tag_hash(word));
```

FIGURE 2-2 Start-State-Tagger

In the preparation stage, the program reads all the files and creates appropriate tables and arrays. Hash tables are used to minimize search time. For each line read from LEXICON, the program registers the most frequent *tag* with the entry name *word* to the lexicon table, referred to as lexicon_hash hereafter. When the lexicon_hash is complete, the program starts reading CORPUS. Each word in the corpus is then looked up in the lexicon_hash. If not found in the lexicon_hash, the word is registered to another table called ntot_hash. If found, the word is discarded. Each BIGRAM is registered to the appropriate hash tables, left_hash and right_hash, only when either the first word is in lexicon_hash and the second word is in ntot_hash, or the first word is in ntot_hash and the second word is in lexicon_hash. The rules read from LEXICALURULES are read and stored in a dynamic array for faster access, since the rules will be applied in order. After all the tables and the array are prepared, the words registered in ntot_hash, which are the words not found in LEXICON, are emptied into a dynamic array called tag_array_key, which has a corresponding array called tag_array_value. The two arrays are related so that the tag for the word in the nth element of tag_array_key is found in the nth element of tag_array_value. After all the arrays and registers are completed, the tagger is ready to find the most likely tag for each word not found in LEXICON.

The rule application step finds the most likely tag for words not found in LEXICON. This procedure is simple and straightforward. Each element of tag_array_key is initially tagged as NN or NNP depending on the case of its first letter. When applying rules, for each rule, every element in tag_array_key is tested for the condition of the rule and, if it matches, the corresponding tag in tag_array_value is updated. The rules are applied in order, so that if there are multiple rules whose condition match a word, the last rule examined decides the tag. Some examples of LEXICALRULE are as follows. For the description of each tag[2], refer to table 2 in Appendix A.

2 The tag set is from the University of Pennsylvania Treebank Tag-set.

- **NN s fhassuf 1 NNS**
  Change the tag from NN to NNS if the word has the suffix ***s***

- **ed hassuf 2 VBN**
  Change the tag to VBN if the suffix is ***ed*** regardless of current tag

- **ly addsuf 2 JJ**
  Change the tag to JJ if adding the suffix ***ly*** results in a word

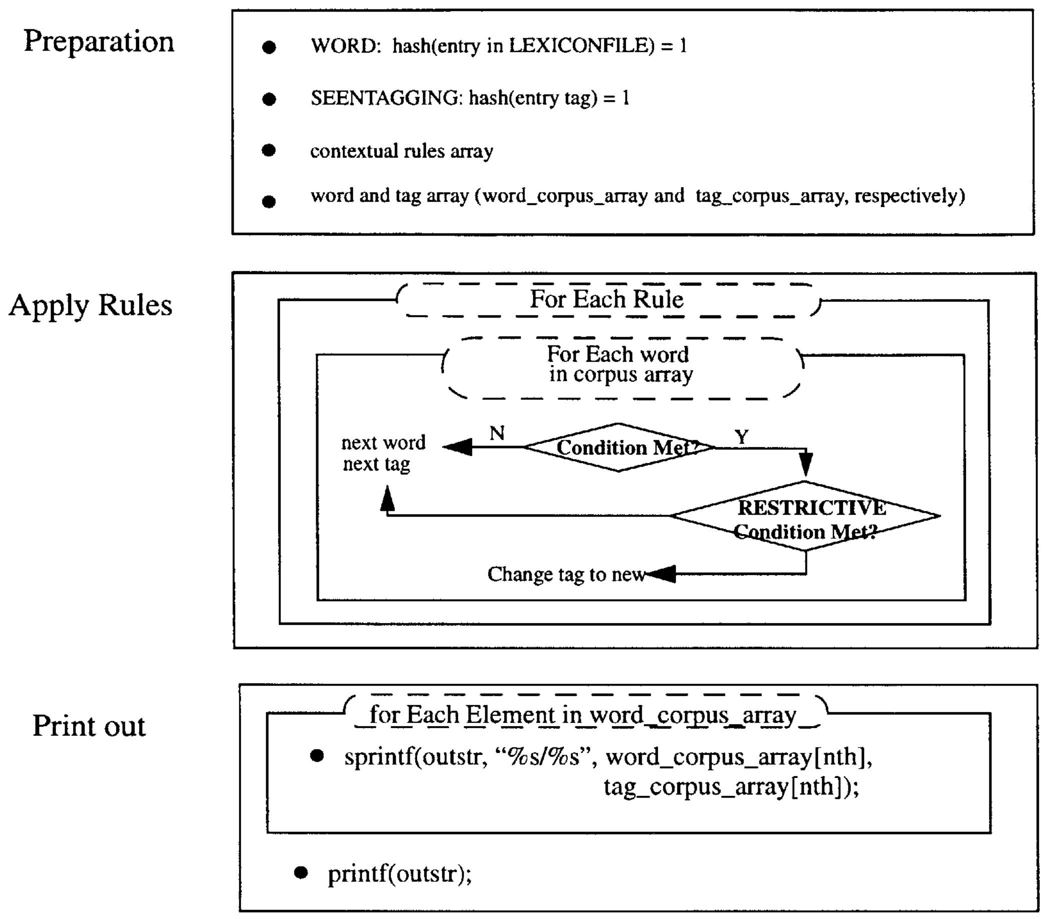
- **un deletepref 2 JJ**
  Change the tag to JJ if deleting the prefix ***un*** results in a word

When every element of tag_array_key have been tested with all the rules, the words in tag_array_key and their corresponding tags in tag_array_value are registered to a newly created hash table, called tag_hash, which has the same format of lexicon_hash. Since every element of tag_array_key is a word not found in lexicon_hash, lexicon_hash and tag_hash are mutually exclusive, and collectively exhaustive.

The output is generated by reading the words from CORPUS and finding the tag for each word from either lexicon_hash or tag_hash. The system prints the word and tag to stdout in the format "word/tag."

2.2.2 The final-state-tagger

The final-state-tagger reads the tagged corpus from stdin and improves the tag accuracy using contextual transformation. It requires five arguments: CONTEXTUAL RULEFILE, LEXICONFILE, corpus size, number of entries in LEXICONFILE, and number of tags found in LEXICONFILE.


Preparation
WORD: hash(entry in LEXICONFILE) = 1
SEENTAGGING: hash(entry tag) = 1
contextual rules array
word and tag array (word_corpus_array and tag_corpus_array, respectively)
Apply Rules
For Each Rule
For Each word in corpus array
next word
next tag
N
Condition Met?
Y
RESTRICTIVE Condition Met?
Change tag to new
Print out
for Each Element in word_corpus_array

```
sprintf(outstr, "%s/%s", word_corpus_array[nth],
                         tag_corpus_array[nth]);
printf(outstr);
```

FIGURE 2-3 The final-state-tagger

CONTEXTUAL RULEFILE is read and used for contextual transformation. The corpus size is needed to allocate arrays for word_corpus_array and tag_corpus_array. Every word in corpus is copied to word_corpus_array and the corresponding tag is stored in tag_corpus_array of the same index. LEXICONFILE is the same file used for initial tagging in the start-state-tagger. LEXICONFILE, the number of entries, and the number of tags are all used only for the RESTRICT_MOVE mode. In this mode, the rule of "change a tag from x to y" will only apply to a word if:

- the word was not in the training set or

- the word was tagged with y at least once in the training set

When training on a very small corpus, better performance might be obtained by not using this mode and setting preprocessor definition RESTRICTIVE_MOVE to 0, since not many possibilities of tags would have been encountered, the contextual rules may be effective to find correct tags that have not been encountered. In our application, the RESTRICTIVE_MOVE mode is used to prevent side effects, as in the following examples:

a) I/PRP am/VBZ going/VBG to/TO hospital/NN.

The sentences are initially tagged in the start-state-tagger. Since, in the start-state-tagger, every word is annotated only with its most frequent part-of-speech, *to* is always annotated as TO. It is the final-state-tagger's job to resolve the contention among the most likely part-of-speech and other possible parts-of-speech.

Suppose that there exists a contextual rule for preposition *to*. Since prepositions come before nouns, one plausible rule may be,

TO IN NEXTTAG NN (change TO to IN if the next word is tagged as NN).

Suppose further that there is another rule to identify verb that comes after infinitive *to*. Suppose the rule states

NN VB PRETAG TO(change NN to VB if the previous word is tagged as TO).

If RESTRICTIVE_MOVE mode is not used, the final-state-tagger will blindly apply the rules. The outcome will differ depending on which rule was applied first. If the rule of TO IN NEXTTAG NN is applied first, the annotation **b** will result. If the other rule is applied first, the annotation **c** will result.

> **b)** I/PRP am/VBZ going/VBG to/IN hospital/NN.
> **c)** I/PRP am/VBZ going/VBG to/TO hospital/VB.

However, if the program runs in the RESTRICTIVE_MOVE mode, the tag of *hospital* can change to VB only when *hospital* was found to be tagged as VB in the training set. Since the only possible tag of *hospital* is NN, it won't be changed to VB even if both rules are found in the rule table. In RESTRICTIVE_MOVE mode, the only possible annotation is **c**.

The final-state-tagger works in three stages as in the start-state-tagger. The final-state-tagger starts by preparing hash tables and arrays. If the program runs in the RESTRICTIVE_MOVE mode, it creates two hash tables: WORDS and SEENTAGGING. The program registers all the entries found in LEXICONFILE to WORD, and all the tagging seen with the entries in LEXICONFILE to SEENTAGGING. The sizes of the tables are initialized with the number of entries and the number of tags, which are provided to the program as arguments. The transformation rules are read from CONTEXTUALRULEFILE to a dynamic array called rule_array. The corpus, which is the output of the start-state-tagger, is read from the stdin until the EOF is encountered, and each word and tag are stored in word_corpus_array and tag_corpus_array, respectively. The size of the arrays are the corpus size, provided as an argument to the program.

In the rule application stage, each element in word_corpus_array is tested for each rule. If condition matches, the candidate tag is stored in *new*. The corresponding tag of the element is changed to *new* if one of the three following condition meets.

- The program does not run in the RESTRICTIVE_MOVE mode,
- the program runs in the RESTRICTIVE_MOVE mode and the word is unknown, i.e. not registered to WORD, or
- the program runs in the RESTRICTIVE_MOVE mode, the word is registered in WORD and the *new* is registered in SEENTAGGING.

After applying all the rules, the final-state-tagger constructs the output by placing each element of word_corpus_array and tag_corpus_array in the format *word/ tag*. The output is printed to stdout.

Figure 2-3 shows the abstract representation of the final-state-tagger.

2.3 Training Transformation-based Part-of-speech Tagger

2.3.1 Error-Driven Learning

The stochastic tagger, based on the probabilities of empirical results, achieves high tagging accuracy, with information contained in a large number of contextual probabilities and the result of multiplying different combinations of these probabilities together. However, the transformation-based tagger system uses corpus-based error-driven learning algorithm that captures linguistic information in a set of simple contextual rules.

Transformation-based error-driven learning works in an iterative process by applying rules learned to annotate text, comparing the result with the truth, and deriving a list of rules that can be applied to the text to make it better resemble the truth which, used as the reference, is a manually-annotated corpus. Thus, to define a specific application of transformation-based learning, one must specify

- the initial state annotator
- the space of allowable transformations
- the objective function for comparing the corpus to the truth and choosing a transformation

The unannotated text is first passed to the initial-state annotator, which can range in complexity, doing anything from assigning random structure, like labelling all the words naively as nouns, to doing the work of a sophisticated annotator which labels all words with their most likely tag as indicated in the training corpus. Once the text passes through the annotator, the text is compared to the manually-annotated text. A list of rules that scores the highest is learned and applied. The iteration goes on until no more rules with higher score than specified can be obtained[9][10].

The transformation rules learned are composed of two parts: a rewrite rule and a triggering environment. Taken together, each rule states *change x to y*(rewrite) *when*(triggering environment).

2.3.2 Evaluation of the Tagger on MUC-II Data

To find out the performance of the transformation-based error-driven learning tagger system for our application, the system was trained and evaluated with respect to the MUCII data corpus, our translation domain. MUCII is an acronym for the Second Message Understanding Conference. The MUCII data corpus includes a number of samples of naval operation transcripts. Due to the nature of operations, the messages are highly elliptical, and contain many incomplete and run-on sentences.

Some examples of MUC-II sentences are as follows.

- spencer lock on to contact with fire control radar and initiated radio communication in unsuccessful attempt to identify.
- estimate heavy damage.
- vid confirms badgers unarmed.

The messages are equivalent to

- spencer lock*ed* on to *the* contact with *her* fire control radar and initiated radio communication in *an* unsuccessful attempt to identify *it.*
- *spencer* estimate*d there was* heavy damage.
- visual id confirms *the* badgers *were* unarmed.

As noted in above examples, MUC-II sentences often leave out articles, link verbs, etc.. In order to assess the influence of such omissions, it is important to train the rule-based tagger on the MUC-II domain and evaluate its performance.

In order to train the rule based tagger system on the MUCII Data, the corpus was divided into three parts, which we can call sets A, B, and C, respectively. Sets A and B were manually annotated to generate a lexicon and transformational rules. Set C was used for three types of evaluation. Set C was annotated with the tagger system with

1. the lexicon and transformational rules acquired from the BROWN CORPUS and the WALL STREET JOURNAL CORPUS.
2. the lexicon and transformational rules acquired from the training on data sets A and B.
3. the lexicon obtained from the BROWN CORPUS, the WALL STREET JOURNAL, and data sets A and B; and transformational rules acquired from training on data sets A and B.

The evaluation result is summarized as follows.

TABLE 2-1. Evaluation on MUCII Data

Evaluation	Total Words	Mistagged	Percentage of Mistagged
1.	2753	267	9.7
2.	2753	127	4.6
3.	2753	125	4.5

As shown in the table above, the tagger system performs reasonably well in the MUCII data domain. Most of the mistagging were effects of the highly elliptical nature of the sentences in the MUCII data, where the link verb *be* and the auxiliary verbs *have* are often omitted. Note that most previous experiments have been performed on more natural English texts. The tagger often mistagged when distinguishing between a past verb and a past participle verb.

Chapter 3

Integration of Transformation-based Part-of-Speech Tagger

3.1 Introduction

A primary concern when constructing multi-module application is coordinating the behavior of each module with others. Coordination is achieved through the means of interface and synchronization. Poorly suited interfaces between two systems reduces efficiency of the application. Poorly designed synchronization is as disastrous if not worse. The application may not guarantee correct operation in most of the times without well designed synchronization.

In the integration of the tagger into the translation system, our primary concern is the interface between the two systems. The system interface between start-state-tagger and final-state-tagger is customized to the processing of text files containing many sentences, and is inefficient in our primary target application, real time speech translation system. The tagger system is organized such that each system, start-state-tagger and final-state-tagger, terminates upon completion of processing the corpus. Since each system needs to load tables of large sizes which involve reading and processing large files such as LEXICON.initially before processing the corpus, activation of the tagger system for a sentence that contains unknown word or construction is inefficient.

In order to provide a interface between the start-state-tagger and final-state-tagger for efficient processing of sentences of unknown words and constructions, an attempt at utilizing a TCP/IP socket interface was made. TCP/IP provides reliable connections, and stream like interface. The synchronization is done by blocked read or write operation. With blocked read/write operation the system is blocked until the system receives desired messages back.

The same method of interface is utilized between the tagger system and the translation system. The desired tagger should work as a background process to the translation system. The tagger reads in necessary files and build tables when initialized. The tagger carries out the infinite loop of receiving input sentences from the translation system, processing and giving back the result of tagging to the translation system. The tagger can interface with the translation system using an appointed temporary file. However, a pipe, a widely used interprocess communication stream, provides a robust and fast means of communication with easier synchronization between the two systems.

The part-of-speech tagger implemented as a server, receives a sentence and returns the sentence with each words tagged with its most likely part of speech in the sentence. The translation system then prepares the tagged sentence into a format legitimate as the input to the understanding system, the grammar for which has been adapted to handle the input consisting of part-of-speech sequences. Then the understanding system tries the parsing of the part-of-speech sequence corresponding to the input word sequence on the basis of the augmented grammar rules.

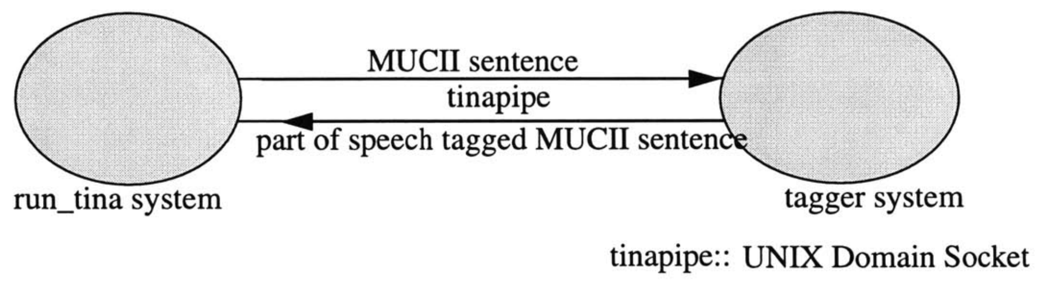

MUCII sentence
tinapipe
part of speech tagged MUCII sentence
run_tina system
tagger system
tinapipe:: UNIX Domain Socket

FIGURE 3-1 The interface between the two system using UNIX Domain Socket

3.2 A Multiprocess Tagger System

The tagger system which is comprised of two programs used pipelined stdout and stdin as the interface. This requires one program to terminate so that the stdout stream is closed and return EOF when read operation is tried on the stream. This interface between the two programs optimizes the system performance with modestly sized corpus. However, for the purpose of our application where the system has to process a sentence at a time, it is at our best interest to keep both programs running and provide a interprocess stream other than stdin/out. A socket, a kind of stream between processes, provides a robust means of interface between the two processes. Figure 3-1 shows the interface, and Figure 3-2 shows the multiprocessor tagger with modified processes. The two processes are named after its predecessor with prefix ll_, in order to denote the modification.

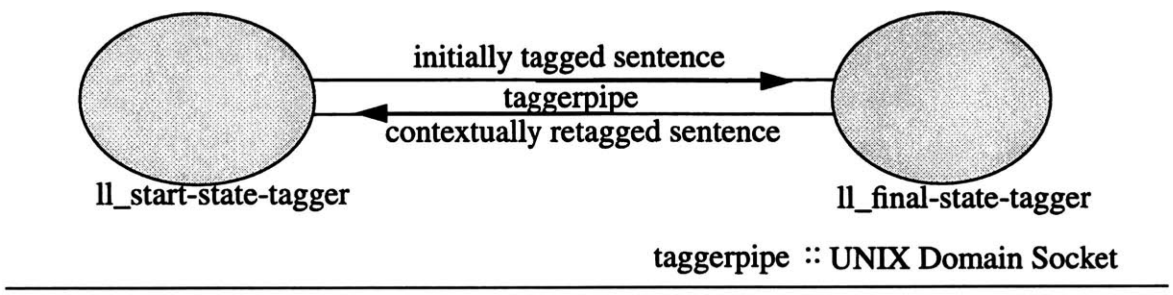

initially tagged sentence
taggerpipe
contextually retagged sentence
ll_start-state-tagger
ll_final-state-tagger
taggerpipe :: UNIX Domain Socket

FIGURE 3-2 Multiprocess Tagger

Each process resembles its predecessor in functionality. ll_start-state-tagger generates initially tagged sentence either with the most frequently occurring tag associated with the word, or assigned tag based on cues such as prefix, suffix, etc. ll_final-state-tagger then updates the tag with contextual transformations. However, to support the new interface, each program is modified accordingly. The system mimics server/client application, and ll_final-state-tagger plays the role of server, since ll_start-state-tagger consults ll_final-sate-tagger after it initially annotate each sentence.

ll_start-state-tagger also works in three stages. In preparation, it first forks, and creates another process just like it, which will replaced by ll_final-state-tagger with system call. If it succeeds in forking and ll_final-state-tagger process has started successfully, the ll_start-state-tagger process establishes an UNIX Domain socket connection to taggerpipe, which has been setup by ll_final-state-tagger. Initializing the hash tables follows the communication setup. Most tables are created with the same method as its predecessor's. Also, for the start-up tag_hash, the start-up corpus is processed once under the same method with start-state-tagger. For processing each line, the process now executes infinite loop of getting sentence from stdin, applying rules, write the intermediate result on the taggerpipe, reading final tagged sentence from the taggerpipe, and writing it out to stdout.

Since ll_final-state-tagger plays the role of a server, it starts from setting up the UNIX Domain socket, /tmp/taggerpipe, that will be shared with ll_start-state-tagger. After ll_start-state-tagger makes the connection, ll_final-state-tagger go through the same preparation stage, reading tables, and rules. Then it goes into a infinite loop of receiving a tagged sentence from ll_start-state-tagger via /tmp/taggerpipe, applying rules, and sending the final tagged sentence back to ll_start-state-tagger by writing it onto /tmp/taggerpipe.

Operating in a multiprocess mode requires a little bit more attention to the error handling. If appropriate error handling has not been provided, and one of the processes exits abnormally, the other process stays on the CPU unproductive unless manually killed. This is the motivation to have the ll_final-state-tagger process as the child of the other process. With parent-child process model, each process has access to the process ID number of the other process, and has more leverage on announcing its status to the other. For example in our application, the signals defined in <signal.h> have been utilized for emergency announcement. Each process send SIGINT to the other process, and to itself when any failure is returned from reading and writing on sockets. Each process implemented with SIGINT handler, then take an appropriate action to the event, such as removing /tagger/pipe.

3.3 Integration with the Translation System

So far, we have discussed the client/server model of the multi-process tagger using stdin/stdout as the user interface. In order to be integrated with the translation system, it requires just a little bit of modification of the tagger discussed above, since it is only change of user from human operator to the translation system. The multi-process tagger becomes ultimately a server to the translation system, by providing services if tagging a sentence to the translation system when requested. Hence, modification of the interface stream suffices the purpose. Again, an UNIX Domain socket, /tmp/tinapipe, has been used to provide the stream between the translation system and the tagger.

Chapter 4
Conclusion

An effort to increase the robustness of a machine translation system has been made. Advantages and problems with highly lexicalized grammar rules have been identified, and a technique utilizing grammar rules based on part-of-speech tagging has been studied, and proven effective. Therefore this work was directed at integrating a part-of-speech tagger with the machine translation system.

A transformation-based error-driven learning tagger has displayed the capability to be trained in our translation domain effectively. The tagger has been adapted to the robust translation system.

The system interface within the tagger system is customized toward processing of text files containing multiple sentences, and was inefficient for the ultimate application of our translation system, real time speech translation, where our system needs the tagger to efficiently process one sentence at a time.

A more efficient interface has been implemented utilizing TCP/IP socket connections. The system was modified around the interface. The tagger system which receives requests from the translation system when the translation system encounters sentences containing unknown words or constructions.

The new implementation achieves efficiency for sentence-by-sentence processing by eliminating the need to initialize large tables for each sentence. All table initialization is done once at start-up, and therefore there is no need to re-initialize the tables for each sentence.

Chapter 5 Future Direction

Since this work was completed, the Group at Lincoln Laboratory has continued to utilize the part-of-speech tagger with substantial success.

Future work might include extending this approach to translation of other languages, for example, to Korean-to-English translation. This would require both a Korean understanding system and a Korean part-of-speech tagger.

Appendix A

TABLE A-1. Description of Tags Used in Examples

Tag	Description
NN	noun, common, singular or mass
NNS	noun, common, plural
IN	preposition or conjunction, subordinating
JJ	adjective or numeral, ordinal
PRP	pronoun, possessive
VBZ	verb, present tense, 3rd person singular
VBG	verb, present participle, or gerund
VBN	verb, past participle
VBD	verb, past tense

Bibliography

[1] Dinesh Tummala, Stephanie Seneff, Douglas Paul, Clifford Weinstein, Dennis Yang,*CCLINC: System Architecture and Concept Demonstration of Speech-to-Speech Translation for Limited-Domain Multilingual Applications*, Proceedings of the 1995 ARPA Spoken Language Technology Workshop, Austin, TX, January 1995, pp.227-232.

[2] Dennis W. Yang, *Korean Language Generation in An Interlingua-Based Speech Translation System*, M.Eng. Thesis, MIT, Cambridge, MA, 1995.

[3] Weinstein, C. J., Tummala, D., Lee, Y. S., and Seneff, S., *Automatic English-to-Korean Text Translation of Telegraphic Messages in a Limited Domain*, International on Computational Linguistics 1996, Copenhagen, Denmark. Also published in C-STAR II Proceedings of the Workshop, ATR International Workshop on Speech Translation, September 1996, Kyoto, Japan

[4] Lee, Y. S., Weinstein, C. J., Seneff, S., and Tummala, D., *Ambiguity Resolution for Machine Translation of Telegraphic Messages*, Proceedings of the Association for Computational Linguistics, Madrid, Spain, July 1997.

[5] Weinstein, C. J., Lee, Y. S., Seneff, S., Tummala, D. R., Carlson, B., Lynch, J. T., Hwang, J. T., and Kukolich, L. C., *Automated English/Korean Translation for Enhanced Coalition Communications*, Lincoln Laboratory Journal Volume 10, Number 1, 1997.

[6] Stephanie Seneff, *TINA: A Natural Language System for Spoken Language Applications*, Computation Linguistics, 18:1, pages 61-88, 1992.

[7] James Glass, Joseph Polifroni, Stephanie Seneff, *Multilingual Language Generation Across Multiple Domains*, International Conference on Spoken Language Processing, Yokohama, Japan, September 1994.

[8] E. Brill, *A simple Rule-Based Part of Speech Tagger*, Proceedings of the Third Conference on Applied Natural Language Processing, ACL, Trento, Italy, 1992.

[9] E. Brill, *A Report of Recent Progress in Transformation-Based Error-Driven Learning*, Ms. Spoken Language Systems Group, Laboratory for Computer Science, MIT.

[10] E. Brill, *Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging*. Computational Linguistics, Dec 1995.

[11] Kenneth Ward Church, *A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text*, Proceedings of the Second ACL Conference on Applied Natural Language Processing, Austin, TX, 1988.

[12] Thompson, A., *Maximizing Reuse During Reengineering*, Proceedings of the Third International Conference on Software Reuse: Advances in Software Reusability, Rio de Janeiro, Brazil, 1994