

TJHSST Computer Systems Lab Senior  
Research Project  
Development of a German-English Translator  
2007-2008

Felix Zhang

June 10, 2008

**Abstract**

Machine language translation as it stands today relies primarily on rule-based methods, which use a direct dictionary translation and at best attempts to rearrange the words in a sentence to follow the translation language's grammar rules to allow for better parsing on the part of the user. This project seeks to implement a rule-based translation from German to English, for users who are only fluent in one of the languages. For more flexibility, the program will implement limited statistical techniques to determine part of speech and morphological information.

**Keywords:** computational linguistics, machine translation

## 1 Introduction

A perfect machine translation of one language to another has never been achieved, because not all language expressions used by humans are grammatically perfect. It is also infeasible experimentally to code in every single grammar rule of a language. However, even a basic program that translates the basic idea of a sentence is helpful for understanding a text in a given language.

## 1.1 Scope of Study

I will focus on a rule-based translation system, because of time and resource constraints. I will start with part of speech tagging and lemmatization, and then progress to coding in actual grammar rules so that sentences can be parsed correctly, so that my program can handle more complex sentences as I embed more rules. I will also expand the program to incorporate limited statistical methods, including part of speech tagging and linguistic property tagging. At best, the program should be able to translate virtually any grammatically correct sentence, and find some way to resolve ambiguities.

## 1.2 Purpose

The goal of my project is to use rule-based methods input to provide a translation from German in to English, or vice versa, for users who only speak one of the languages. Though the translation may be simple, the program still aids a user in that it provides a grammatically correct translation, which facilitates understanding of even primitive translations. Basic translations of short passages are especially helpful for users reading less formal text, as sentence structures tend to be less complex.

# 2 Background and review of current literature and research

Rule-based translation is the oldest form of language processing. A bilingual dictionary is required for word-for-word lookup, and grammar rules for both the original and target language must be hard coded in to structure the output sentence and create a grammatical translation. Most online translators currently are based off of SYSTRAN, a commercial rule-based translation system.

The more modern technique, statistical machine translation, is the most-studied branch of computational linguistics, but also the hardest to implement. Statistical methods require a parallel bilingual corpus, which the program reads to "learn" the language, determining the probability that a word translates to something in a certain context using Bayes' Theorem:

They can also be used to determine linguistic properties, such as part-of-speech and tense. Usually, statistical methods are more accurate when

Figure 1: Bayes' Theorem.

$$\tilde{e} = \mathit{arg} \max_{e \in e^*} p(e|f) = \mathit{arg} \max_{e \in e^*} p(f|e)p(e)$$

the corpus used is larger (Germann, 2001). Statistical methods are considerably more flexible than rule-based translation, because they are essentially language-independent. Google Translate, which has access to several terabytes of text data for training, currently is developing beta versions of Arabic and Chinese translators based on statistical methods. Most research is being done with much more funding and resources than my project, and is thus much more advanced than my scope.

Statistical methods can also be implemented in a trivial manner, in which the tag, translation, or other linguistic component of a word is not based on context, but only how frequently the tag is associated with the given word in a corpus, regardless of context or other factors. This simpler implementation will be used in my program.

### 3 Development

The main components to a rule-based translator are a bilingual dictionary, a part of speech tagger, a morphological analyzer that can identify linguistic properties of words, a lemmatizer to break a word down to its root, an inflection tool, and a parse tree.

#### 3.1 Dictionary

The dictionary stores a German word, its part of speech, its English translation, and any other data relevant to its part of speech, for example, for nouns, it also lists its plural form and gender. A large dictionary would be impractical for testing purposes, so I only include pronoun forms, conjunctions, and articles, with only a few nouns and verbs. These entries are stored in a hashtable, with German words as keys and English translations as values.

## 3.2 Part of speech tagging

The program first attempts to tag words in the input sentence using the freely available TIGER corpus, which consists of 700,000 German tokens, with each token manually assigned a part of speech. For large, full sentences, the program stores the entire corpus into a hashtable. Each unique word in the corpus serves as a key, while each table value is a list of tuples. Each tuple represents a different part of speech assigned to the word in the corpus. The first element in the tuple is the part of speech, while the second is a number, indicating the frequency of the tag's occurrence. For single words and short phrases, it is more efficient to search for the single word in the corpus, and incrementing a separate counter for the occurrences of each different part of speech assigned to it. When a word, usually a noun or verb, is unable to be looked up in the corpus, a rule-based system is used as backoff. These rules are specific to the language being translated. For example, if a word is in between an article and a noun, it will be tagged as an adjective.

## 3.3 Morphological Analysis

Morphological analysis would use definite articles, suffixes, and adjective endings to determine linguistic properties such as gender, case, tense and person. It generates possible pairs of gender and case for nouns, and tense and conjugation for verbs. Two separate sets of pairs are generated for articles and modifiers, and the final list of possibilities is derived from the intersection of these two sets. To reduce ambiguity, a method for noun-verb agreement is used to determine the subject of the sentence. This information is used for lemmatization.

Morphological analysis can also be implemented statistically. Since each token in the TIGER Corpus is also assigned linguistic information such as gender, case, and number, the likelihood of a word having certain linguistic properties can be calculated. The simplest calculation would be for gender, since singular words will not change gender in different contexts.

## 3.4 Noun-phrase chunking

The purpose of noun-phrase chunking is to collate words in a sentence which would group together to form a sentence element, such as the subject. A sentence element will typically consist of more than just a single word. Not

only the noun, but also any articles and modifiers are included, such as “the large man”, instead of just “man”. The program searches for nouns in the sentence, and finds the closest modifiers and articles to group into a “chunk”, which is later identified as a specific sentence element.

### **3.5 Noun-verb agreement**

Since each word will often generate several different possibilities during morphological analysis, a method for noun-verb agreement is used to reduce ambiguities. The properties of the nouns nearest to the verb in the sentence are crosschecked with the properties of the verb, according to conjugation. A singular noun, if next to a singular third-person verb, will most likely be the subject of the sentence. If two nouns in the sentence match the verb, the first one is taken as the subject, as this word order is more common. Once the subject has been determined, the program removes the possibility that any other nouns could be the subject. This method helps to disambiguate verbs and nouns, by reducing the possibilities of gender, case, tense, and person. In testing, this method helped to reduce ambiguities to about one per word.

### **3.6 Lemmatizer**

The lemmatizer takes information from the morphological analysis and breaks a word down into its root form. For nouns, this means that plural nouns should be reduced to singular form, and suffixes resulting from different grammatical cases should be removed. When the program encounters a word that may be plural, it attempts to remove any of the common verb endings from the word: -e, -en, -er, -ern, and -s. For verbs, any ending from conjugation or tense should be removed. The program takes the few possible conjugation endings, "-e", "-st", "-t", and "-en", removes them, and adds "-en" to the root to render the infinitive form of the word. The prefix for past-tense verbs, "ge-", is also searched for and removed. This saves considerable space in the dictionary, as I do not have to code in every inflected form of every word.

### **3.7 Parse tree**

The most rudimentary form of this method comes in two parts. First, the program, given phrase chunks with the linguistic properties of the noun or verb, assigns the chunk to a specific sentence element. For example, nouns

that are in the nominative case, regardless of number, will always be subjects, accusative nouns will always be direct objects, and verbs in the present tense will be main verbs. The program then assigns a priority number, based on where the sentence element normally would occur in an English sentence. For example, the subject will come before the main verb and the direct objects, and have a priority of 1, while the indirect object comes at the end, with a priority of 5. A more advanced version of the parse tree arranges the sentence based on dependency grammar. Verbs connect from the subject to the direct object, and articles and adjectives are nodes of nouns. In translation, this tree must be rearranged to accommodate the target language's grammar.

### **3.8 Inflection**

Since the dictionary lookup will only produce the root form of the translated word, a simple inflection tool is used to conjugate words, once translated into English. Inflection requires the information from the morphological analysis, which it then uses to add endings to words. Words marked as plural add an "-s" or "-es" to the end, as do singular verbs, depending on whether the root word ends in a consonant or a vowel. Also taken into account are common ending changes, such as words ending in "-y" turning into "-ies" in the plural, and past tense endings for weak verbs, which always follow a pattern of adding "-ed" to the ending. In addition to conjugation, the method also capitalizes the first word of each sentence, and adds a period to the last word of the sentence, assuming the sentence is a statement.

### **3.9 Implementation of Statistical Methods**

Part of speech tagging and morphological analysis were chosen for statistical implementation, since the TIGER Corpus readily came with the information for each word. The program creates a hashtable, with each unique word appearing in the corpus as a key, and all tags associated with the word, along with a frequency count, as the value.

## 4 Testing

### 4.1 Rule-based Methods

Testing is conducted through input of sentences with new features. To test my lemmatizing component, I would input various inflected forms of a word to check the uniformity of the program's output. Varying sentence structures can also serve as a functional test to check the validity of newly coded grammar rules in the parse tree.

### 4.2 Statistical Methods

Both statistical methods are tested using similar techniques. As the program goes through each word in the corpus, it checks the tag that the program determines is most likely to be assigned against the one that appears on the line with the word. The program maintains a count of the "matches" that are made, which it uses to determine a percent accuracy.

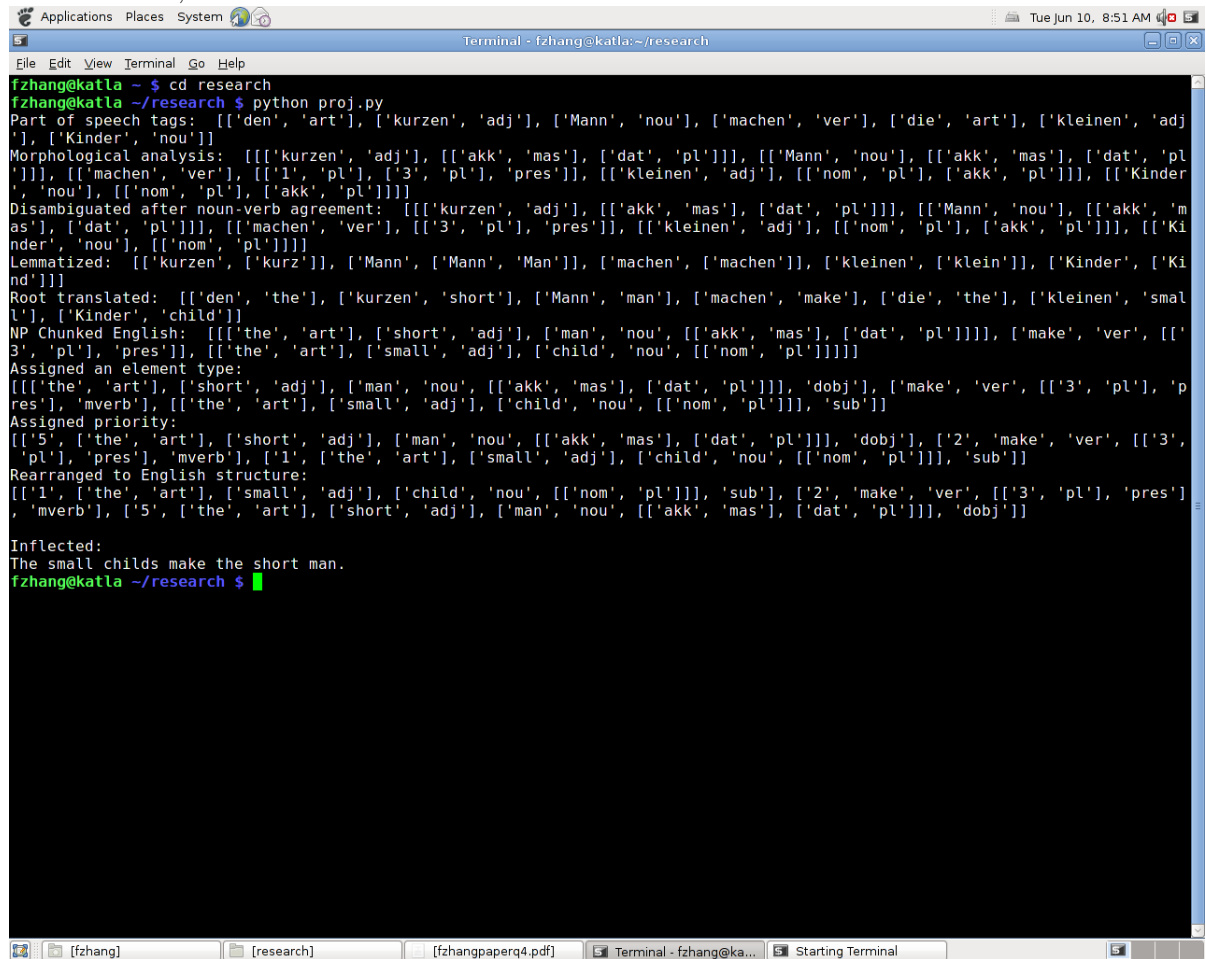
## 5 Results

My program is able to translate a simple German or English sentence into the other language, provided the word is known in the lexicon. A statistical tagger correctly resolves most ambiguities in words. The project fulfills its purpose as a simple translator with basic grammar rules and basic statistical techniques, but would need an implementation of more advanced statistical methods to attain more flexibility in sentence structure parsing.

### 5.1 Word Ambiguity

In German, many words can be taken to very different meanings depending on the contexts. For example, the German pronoun "sie" can be translated to "she", "her", "they", "them", or "you". Though the program does attempt to resolve as many ambiguities as possible using noun-verb agreement, there still exist cases wherein even a native human speaker of German would have trouble disambiguating, such as a sentence in which both nouns could possibly be the subject.

Figure 2: The rule-based translation component, running a translation on the sentence, “den Mann machen die kleinen Kinder.”



```

fzhang@katla ~ $ cd research
fzhang@katla ~/research $ python proj.py
Part of speech tags: [['den', 'art'], ['kurzen', 'adj'], ['Mann', 'nou'], ['machen', 'ver'], ['die', 'art'], ['kleinen', 'adj'], ['Kinder', 'nou']]
Morphological analysis: [['kurzen', 'adj'], [['akk', 'mas'], ['dat', 'pl']], [['Mann', 'nou'], [['akk', 'mas'], ['dat', 'pl']], [['machen', 'ver'], [['1', 'pl'], ['3', 'pl'], 'pres']], [['kleinen', 'adj'], [['nom', 'pl'], ['akk', 'pl']], [['Kinder', 'nou'], [['nom', 'pl'], ['akk', 'pl']]]
Disambiguated after noun-verb agreement: [['kurzen', 'adj'], [['akk', 'mas'], ['dat', 'pl']], [['Mann', 'nou'], [['akk', 'mas'], ['dat', 'pl']], [['machen', 'ver'], [['3', 'pl'], 'pres']], [['kleinen', 'adj'], [['nom', 'pl'], ['akk', 'pl']], [['Kinder', 'nou'], [['nom', 'pl']]]
Lemmatized: [['kurzen', ['kurz']], ['Mann', ['Mann', 'Man']], ['machen', ['machen']], ['kleinen', ['klein']], ['Kinder', ['Kind']]]
Root translated: [['den', 'the'], ['kurzen', 'short'], ['Mann', 'man'], ['machen', 'make'], ['die', 'the'], ['kleinen', 'small'], ['Kinder', 'child']]
NP Chunked English: [['the', 'art'], ['short', 'adj'], ['man', 'nou'], [['akk', 'mas'], ['dat', 'pl']], ['make', 'ver'], [['3', 'pl'], 'pres']], [['the', 'art'], ['small', 'adj'], ['child', 'nou'], [['nom', 'pl']]]
Assigned an element type:
[[['the', 'art'], ['short', 'adj'], ['man', 'nou'], [['akk', 'mas'], ['dat', 'pl']], ['dobj'], ['make', 'ver'], [['3', 'pl'], 'pres'], ['mverb'], [['the', 'art'], ['small', 'adj'], ['child', 'nou'], [['nom', 'pl']], 'sub']]
Assigned priority:
[['5', ['the', 'art'], ['short', 'adj'], ['man', 'nou'], [['akk', 'mas'], ['dat', 'pl']], ['dobj'], ['2', 'make', 'ver'], [['3', 'pl'], 'pres'], ['mverb'], ['1', ['the', 'art'], ['small', 'adj'], ['child', 'nou'], [['nom', 'pl']], 'sub']]
Rearranged to English structure:
[['1', ['the', 'art'], ['small', 'adj'], ['child', 'nou'], [['nom', 'pl']], 'sub'], ['2', 'make', 'ver'], [['3', 'pl'], 'pres'], ['mverb'], ['5', ['the', 'art'], ['short', 'adj'], ['man', 'nou'], [['akk', 'mas'], ['dat', 'pl']], 'dobj']]
Inflected:
The small childs make the short man.
fzhang@katla ~/research $

```



## 5.2 Encoding Problems

A characteristic unique to the German language is the use of special characters in its alphabet, such as diacritic marks. Due to program constraints, these characters can not be expressed directly during input, instead substituting them for their closest equivalents: ö is expressed as "oe", while ß is expressed as "ss". An issue with the corpus lay in the corpus compilers' attempt to encode the special characters, which ended up as garbled ASCII code when the corpus was read into the program.

## 5.3 Corpus Size

Though a larger corpus typically allows for greater accuracy in tagging, file size can be a constraint in many cases. The TIGER Corpus, consisting of 700,000 lines, is 42 megabytes in size, making it impractical for web-based or portable use. The amount of time spent by the program while going through the corpus also presents a problem of convenience and efficiency.

## 5.4 Stem Changes

In general, most inflected verbs in German add a suffix, depending on its conjugation - first person singular adds an "-e", second person singular adds an "-st", and third person singular adds "-t". However, for several exceptions in German, the root word itself alters slightly in singular conjugations. For example, the verb "lesen", which means "to read", has a vowel change when conjugated in the third person singular, "er liest", as opposed to the expected "er lest". Only certain verbs follow this rule, which means the program cannot simply change the vowel stem when it encounters such a conjugation, but the verbs that express this quality are too commonly encountered to simply disregard. A way around this problem is to include an indicator in the dictionary entry for the word, noting that the verb is irregularly conjugated.

Similarly, German verbs are divided into "strong" verbs and "weak" verbs. Weak verbs follow a common pattern in the present perfect tense, adding a "ge-" prefix and a "-t" suffix. The program's morphological analysis easily detects weak verbs. Strong verbs, however, have no set pattern when in the past tense, including many vowel changes. For strong verbs, the only way to resolve the problem is by manually including the past tense form for each strong verb in the dictionary.

To a lesser extent, this is also a problem encountered during English inflection. The inflection method of this program results in much overregularization, because not all English verbs follow the simple “-ed” ending - Many also have stem changes, such as “see” to “saw”.

## 5.5 Complexity

Rule-based translation, by definition, is confined to only a defined set of grammatical structures it can parse. In the program’s priority-number based method of parsing, for example, German sentences can only be rearranged in one specific order. Thus, in terms of flexibility, statistical methods are functionally superior, as they are language-independent, and can be trained for virtually any corpus of sufficient size.

## 5.6 Statistical Accuracy

According to Charniak (1997), when assigning part-of-speech statistically, the accuracy of tagging should approach 90 percent when each word is simply assigned its most frequently occurring tag. Running the part-of-speech tagger on the sample corpus confirms this, yielding an accuracy of 87.977% on TIGER’s full 746,660-word corpus. The morphological analyzer does not perform as well, reaching 73.716% accuracy on the full corpus. The morphological analyzer’s decreased accuracy is largely because a word’s linguistic properties, notably grammatical case, are heavily dependent on context.

## 6 Conclusion

In the context of this German translation program, there is no definite way to compare the accuracies of statistical and rule-based machine translation methods. Simple statistical methods yield a relatively high accuracy, given their primitive method of implementation. The seemingly small error rate, however, translates to a number of errors in a large corpus too large to be acceptable for any professional translation program. Likewise, rule-based translation will provide accurate results, but is bound to a much narrower frame of function.

## 7 Recommendations

Given more time, the program's statistical methods can be expanded to more complex algorithms, as context-based methods tend to yield much higher accuracy. Ideally, the entire translation process should be written using statistical methods, as the program becomes effectively language-independent.

## References

- [1] Brants, Thorsten, "TnT: a Statistical Part-of-Speech Tagger", *Applied Natural Language Conferences*, pp. 224-231, 2000
- [2] Chanod, J, and Tapanainen, P, "Tagging French: Comparing a Statistical and a Constraint-Based Method", *Proceedings of the Seventh Conference on European Chapter of the Association for Computational Linguistics*, pp. 149-156, 1995
- [3] Charniak, E, "Statistical Techniques for Natural Language Parsing", *The American Association for Artificial Intelligence*, pp. 33-43, 1997
- [4] Germann, U, "Building a Statistical Machine Translation System from Scratch", *Proceedings of the Workshop on Data-driven Methods in Machine Translation*, pp. 1-8, 2001
- [5] Lezius, W, "A Freely Available Morphological Analyzer, Disambiguator and Context Sensitive Lemmatizer for German", *Proceedings of the COLING-ACL*, pp. 743-748, 1998
- [6] Nallapati, Ramesh, "Capturing Term Dependencies Using a Language Model Based on Sentence Trees", *Center for Intelligent Information Retrieval*, pp. 383-390, 2002
- [7] Schulte, S, "Inducing German Semantic Verb Classes from Purely Syntactic Subcategorisation Information", *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pp.223-230, 2002
- [8] Tiger Corpus, <http://www.ims.uni-stuttgart.de/projekte/TIGER/TIGERCorpus/>

## Appendix: Statistical Method Code

```
#Statistical part of speech tagging - This program tags words based on their most
def main():
    en = []
    de = []
    corpusdict = {}
    de = readcorpus("tiger_release_aug07.export")
    morphotest(de)
    # hashtable = findpos(de)
    # word = raw_input("Enter word: ")
    # morphosingleword(word,de)

    # if word in hashtable.keys():
    # print hashtable[word]

def findmaxprob(list): #finds the most frequently occurring tag
    max = 0
    maxtag = ""
    for x in list:
        if x[1] > max:
            max = x[1]
            maxtag = x[0]
    return maxtag

def possingleword(word,de): #functions the same way as a hashtable, but only store
    pos = [[]]
    for x in de:
        list = x.split('\t')
        if list[0][0] is not "#" and list[0][0] is not "%" and list[0][0] not in "12345678":
            noblanks = removeblanks(list)
            encountered = False

    if word == noblanks[1] or word == noblanks[0]:
        if pos == [[]]:
            pos = [[noblanks[2],1]]
        for y in pos:
```

```

if y != []:
if y[0] == noblanks[2]:
y[1] = y[1] + 1
encountered = True
if encountered == False:
pos.append([noblanks[2], 1])
print word, pos
print "Most likely tag: " + findmaxprob(pos)

def morphosingleword(word,de): #functions the same way as a hashtable, but only st
    pos = [[]]
    for x in de:
        list = x.split('\t')
        if list[0][0] is not "#" and list[0][0] is not "%" and list[0][0]
            noblanks = removeblanks(list)
            encountered = False

        if word == noblanks[0]:
            if pos == [[]]:
                pos = [[noblanks[3],1]]
            for y in pos:
                if y != []:
                    if y[0] == noblanks[3]:
                        y[1] = y[1] + 1
                        encountered = True

            if encountered == False:
                pos.append([noblanks[3], 1])

print word, pos
    print "Common tag: "
mostlikely= findmaxprob(pos).split(".")
print mostlikely[1:len(mostlikely)]

def readcorpus(filename):
s = open(filename).read().split('\n')[:-1]
return s
def findmorph(readin):
list = []
lookup = {}

```

```

for x in readin:
list = x.split('\t')
if list[0][0] in "abcdefghijklmnopqrstuvwxyzaBCDEFGHIJKLMNOPQRSTUVWXYZ":
noblanks = removeblanks(list)
register(lookup, noblanks[0], noblanks[3])
return lookup
def findpos(readin):
list = []
lookup = {}
for x in readin:
list = x.split('\t') #splits line into list
if list[0][0] != "#" and list[0][0] != "%" and list[0][0] not in "1234567890": #g
noblanks = removeblanks(list)
register(lookup, noblanks[1], noblanks[2])
return lookup

def removeblanks(list): #gets rid of blank entries when each line is converted into
newlist = []
for x in list:
if x != "":
newlist.append(x)
return newlist

def register(table, key, value): #registers the German word in the hashtable as a
if key in table:
invalues = False
sets = table[key]
for x in sets:
if x[0] == value:
x[1] = x[1] + 1
invalues = True

if invalues == False:
table[key].append([value,1])
else:
table[key] = [[value,1]]
def morphotest(readin):
hashtable = findmorph(readin)

```

```

matches = 0
x = 0
for y in readin:
list = y.split('\t')
if list[0][0] in "abcdefghijklmnopqrstuvwxyABCDEFGHIJKLMNOPQRSTUVWXYZ":
noblanks = removeblanks(list)
if noblanks[3] == findmaxprob(hashtable[noblanks[0]]):

print "Match", noblanks[3], findmaxprob(hashtable[noblanks[0]]), noblanks[0]
matches = matches + 1
print matches
x= x+1
print "Total matches: ",
print matches
print "Total words: ",
print x
print "Accuracy: ",
print matches * 100.0 / x*1.0,
print "%"

def test(readin):
hashtable = findpos(readin)
matches = 0
x = 0
for y in readin:
list = y.split('\t')
if list[0][0] in "abcdefghijklmnopqrstuvwxyABCDEFGHIJKLMNOPQRSTUVWXYZ":

noblanks = removeblanks(list)
# printlikely[1:len(mostlikely)]
# noblanks[0:3]
# print findmaxprob(hashtable[noblanks[1]]),noblanks[2],noblanks[1]
if noblanks[2] == findmaxprob(hashtable[noblanks[1]]):
print "Match", noblanks[2], findmaxprob(hashtable[noblanks[1]]),noblanks[1]
matches = matches + 1
print matches
x = x+1
# if x == 1000:

```

```
# break
print "Total matches: ",
print matches
print "Total words: ",
print x
print "Accuracy:",
print matches * 100.0 /x*1.0,
print "%"
main()
```

## Appendix 2: Statistical Testing Results

Test size	Part of speech	Morpho. Analysis
100	90.0%	78.0%
1000	87.4%	77.0%
Full	88.0%	73.7%