## I. Abstract

In Natural Language Processing, a lot of time was spent on statistical methods to parse natural languages. I wanted to also take a look at rule-based statistical parsers. Specifically, I looked at the Japanese language and how a rule-based parser might be able to effectively parse Japanese sentences. My parser consists of several steps, and five main files. I did relatively little research, and instead spent my time developing my system. Because I was using rule-based methods, I also used a separate set of tools called the Xerox Finite State Toolkit to create a rule-based morphological analyzer.

## II. Japanese Language

Japanese is a moderately agglutinative language. This means that the words in Japanese consist of fairly recognizable morphemes. Nouns and prepositions are not agglutinative at all, but adjectives and verbs both take on conjugated forms that can be morphologically analyzed and tagged

In Japanese sentences, there are also one-syllable morphemes known as particles. These particles are used to designate the part of speech within the sentence that the word or phrase preceding the particle is taking. In this way, particles are somewhat similar to non-terminal tags, but they are contained within the initial input. This means that tree creation is a lot simpler than many other natural languages, such as English.

One of the challenges when it comes to tree creation is that there is only one real rule for terminals, which is that the verb phrase must come last. Every other phrase+particle set can appear in any order and still be accurate. This means that many rules need to be created to show the relation of any given tagged phrase to the total sentence at hand.

## III. XFST

Xerox Labs, as sponsored by Stanford University, developed their finite state automata tools, xfst and other, as a rule-based natural language processing toolkit. According to Ken Beesley's book, the toolkit was developed specifically for the Stanford Natural Language Processing department, in order to be able to build finite state transducers for natural language emulation. XFST is a regular expression based language, somewhat similar to PERL. It has two main types of files: rule files and lexicon files. Lexicon files detail the general rules of the language and rule files cover exceptions and small letter changes. In general, the toolkit is used to create rule-based morphological analyzers.[1]

Rule-based parsers are useful for languages that have regular language rules that can be finitely represented. They are more accurate than statistical based methods, but massively over-generate, depending on the number of

[1] Beesley, Kenneth R, *Finite State Morphology* (Stanford, CA : CSLI Publications, 2003).

rules that are in place and the amount of ambiguity in the language. It appears that a Japanese rule-based parser has not been attempted before, but statistical methods are in place. One of the more prominent Japanese parsers currently is the "KN Parser", which uses dependency analysis to be able to parse a sentence from the bottom up.[2]

## IV. My Parser

My parser consists of three parts. First, there is the basic tokenizer (JToken.java) which simply takes in an input sentence and outputs a file that can be morphologically analyzed. Then, there is the morphological analyzer, which is made up of two files, the lexicon, and the rule dictionary. This takes in a Japanese input word and tags it according to it's morphological structure. Most words are only tagged as a part of speech, but verbs and adjectives both have several conjugated forms. These two parts of speech account for the majority of excess tags. I covered six conjugations of adjectives and 16 conjugations of verbs.

### *Adjective Conjugations*

Because of the way files are tokenized in Japanese, only one of the two sets of adjectives had conjugated forms. Na-Adjectives, the other type of adjective, are indicated by a particle tail *na*, which is accounted for in the particle

[2] Kurohashi, Sadao & Nagao, Makoto, *KN Parser: Japanese Dependency/Case Structure Analyzer* (Kyoto, Japan: Kyoto University)

tagger. Therefore, all of the adjective conjugation rules deal with the I-Adjective set. Na-adjectives are also included in the lexicon file, but require no special conjugatory rules.

*Verb Conjugations*

Verbs in Japanese are complicated because of the system of honorifics and humble forms. My morphological analyzer can only handle plain and normal forms of verbs, which are the least polite, but the most commonly used in textual representation. There are 5 types of verbs that my morphological parser recognizes, which are consonant verbs, vowel verbs, and three irregular forms: suru, kuru, and desu. Each type of verb has a different set of 12 rules.

Lastly, JParse consists of a CKY algorithm that generates a tree based on the tags assigned by the morphological analyzer. This is contained within the JParse.java file, which in turn, uses the binary and unary files. These files contain an extremely limited number of parse rules, but ones that should account for all legal sentences in Japanese.

**Implementation**

The Java program JParse combines the code written in the xfst programming language with a simplified CKY parser to accommodate for the possible parse trees for the input sentence. For every sentence that is included in the input file, JParse first tokenizes the file, into sentences, then words. Then, each word is run through the morphological analyzer rule set, which also includes the lexicon. If the word is found in the lexicon, any appropriate rules are applied, and the word is returned in tagged form. The final tag in the sentence is that of the basic part of speech, but all tags can be used to generate a proper parse tree.

Then, once every word in a sentence has a generated tag, the sentence is run through the simplified CKY Parser function. This function assigns non-terminals to phrases and words based on the particles contained within the sentence. Then, from there, the program outputs a graphical representation of the tree in array form, with the tagged words appearing at the left hand side of the tree

**Notes on running JParse**

To actually run JParse, one will need to download the xfst toolkit from http://www.stanford.edu/~laurik/fsmbook/home.html. Then, there are a series of steps one must take to receive a result. First, give JToken a sentence in Romanized Japanese. Then, run the command given in the readme file in a

terminal. Finally, run JParse. Ideally, this would be unified into one system, but Macintosh terminals are not very easily run in Java.

## V. Results

The results of JParse are somewhat difficult to analyze, because there is relatively little test data. Of the sample sentences I used, no ambiguity occurred and the parse trees generated were accurate to the rule set. However, there are situations where JParse will be completely ineffective: for example, when a word is not contained within the lexicon file. During such a case, words would need to be manually added. JParse can currently handle over one hundred words, but to be more accurate, the lexicon would need to be expanded exponentially. See **Future Work** for proposals as to how to accomplish that expansion. In general, with words contained within the lexicon, JParse has no ambiguity and correctly morphologically parses words and creates accurate parse trees for the sentences the words are contained within.

## VI. Future Work

Because this is a rule-based parser, the parser is only as complete as its rule set. The rules I created for the morphological analyzer handle the most common conjugated forms, as well as regular and irregular forms, but I did not add enough rules for this to be an accurate representation of the Japanese language. The rule set could very easily be expanded to accommodate new prepositions, verbs, nouns, or adjectives. Japanese is very

much an open language, especially in the adjective, noun, and verb classes, so any additions to the lexicon will make the tagger more accurate.

The Java program that wraps JParse can also be expanded to add words dynamically to add new words instead of manually inputting them. A word guesser could also be added to the implementation of the morphological analyzer, so that words that were not present in the lexicon file could be guessed at. Without one of these two methods, the parser would need to manually expanded every time it encountered a word it did not know. The word guesser technique, unfortunately, creates a lot of ambiguous parses, and therefore is not ideal.

JParse could also be expanded to take in real Japanese input. This would prove more difficult than taking in Romanized Japanese (*romaji*) because of the way the Japanese language is structured. Word breaks in Japanese are very implicit rather than explicit and training a computer to be able to tokenize Japanese sentences would take a lot of effort. However, there is a tokenizer called JUMAN available for the Japanese language. If it were possible to gain access to this tokenizer, JParse would be able to take in romanized and traditional Japanese sentences.[3]

[3] [3] Kurohashi, Sadao & Nagao, Makoto, *KN Parser: Japanese Dependency/Case Structure Analyzer* (Kyoto, Japan: Kyoto University)

Finally, JParse could find a way to combine rule-based and statistical approaches to finding the correct parse tree. This is most easily implemented by using a probabilistic CFG, but it would also be helpful to have probabilities within the word parses themselves. This is made difficult by the limitations of xfst. Which does not allow probabilities in inherently, but a wrapper program could be created in order to store probabilistic values of the most commonly occurring ambiguities.

It would be interesting to pursue the comparison of a rule-based parser with a statistical parser with the completely unambiguous language, Lojbaan. In the future, I would like to see if with an unambiguous language, gathering the data required to build a statistical model takes less time and effort than that of writing rules that would cover the entirety of that language. This would be unrelated to JParse, but would implement similar techniques.

Bibliography

Beesley, Kenneth R. *Finite State Morphology*. Stanford, CA : CSLI Publications, 2003.

Kurohashi, Sadao & Nagao, Makoto. *KN Parser: Japanese Dependency/Case Structure Analyzer*. Kyoto, Japan: Kyoto University.