Natural Language Processing Paradigms and Applications

Name: Tarik Zemrani
Course: Cs210-AL
Professor: Dr. A. Bellaachia
Date: 04/26/2002

Abstract

Recently, providing computer systems with full human intelligence capabilities has been the main focus of AI researchers. Researchers have made a lot of efforts in creating programs that understand with deep knowledge the semantic meaning of texts. In this paper, I am going to introduce an art that makes the AI researchers dream possible. This art is nothing but Natural Language Processing (NLP). Through this paper, I will introduce the most important NLP paradigms (Rule based, statistical, finite state machine, logic programming and object oriented paradigm). Moreover, I will present a brief overview on the most important tools used in the implementation of NLP applications. These tools include Lexicons such as WordNet and syntactic parsers such as the Link Grammar. Finally, I will provide a design schema that illustrates how lexicons and parsers are used in order to implement a simple NLP application. The application will take a user-defined sentence as input. The application detects the main verb and outputs all the possible synonyms corresponding to that verb.
Table of Contents

1. Introduction................................................................................................................. 3
2. NLP implementation paradigm................................................................................... 3
   2.1. The rule-based paradigm..................................................................................... 3
   2.2. The finite state machine paradigm................................................................. 4
   2.3. The statistical approach................................................................................... 7
   2.4. The Object Oriented Approach....................................................................... 8
   2.5. The processing paradigm............................................................................... 9
   2.6. The Text approach ....................................................................................... 11
3. Main natural language processing tools.................................................................... 12
   3.1. Syntactic parsers (Link Grammar)................................................................. 12
   3.2. Link Grammar Example [7]............................................................................ 13
   3.3. Parsers evaluation methods.......................................................................... 15
   3.4. Lexicons (WordNet) .................................................................................. 17
4. An NLP sample application...................................................................................... 18
   4.1. General Requirements.................................................................................. 18
5. Conclusion................................................................................................................ 20
1. Introduction

NLP is the ability of a program to understand the human natural language. According to Microsoft research group, “The goal of the Natural Language Processing (NLP) is to design and build a computer system that will analyze, understand, and generate languages that humans use naturally, so that eventually you can address your computer as though you were addressing another person.” [1]. Despite all the great improvements that computers has known in the fields of processing speed and computing ability, these dumb machines are still unable to master the basics of humans spoken and written languages.

As an example, an English speaker obviously understands a sentence like “Flying planes can be dangerous”, however, this sentence presents some difficulties to a software program that lacks both the human knowledge of the world and his/her experience with linguistic structures. “Is the best interpretation that the pilot is at risk, or that the danger is to people on the ground? Should the verb "can" be analyzed as a verb or as a noun? Which of the many possible meanings of "plane" is relevant? Depending on context, "plane" could refer to, among other things, an airplane, a geometric object, or a woodworking tool.” [1]. The best solution to this problem is to train the NLP system to learn that a "plane" is a vehicle that flies. The system learns by analyzing existing texts, including online dictionaries and encyclopedias, and automatically acquiring knowledge from this analysis.

2. NLP implementation paradigm

2.1. The rule-based paradigm

Mostly used in applications that detect or verify specific features. The verification is based on a confidence value output by a rule mapping function. The mapping function uses some parameters that can be entered by the user. Mostly, the mapping function is a function of basic properties of the feature to be verified or detected. The NLP system takes action based on a calculated confidence value. A very important advantage of the rule-based paradigm is that rules are modular; they can be added or deleted without affecting the other parts of the system. Moreover, if there is a little training data, the user can use it to fine tune the parameters used in the rule mapping function. (This can improve the ability to produce a better confidence value and therefore, high quality output). [2]. However, The rule-based paradigm is more classical and is not robust since rule based systems can crash in the face of any unexpected input.

As an example, suppose we want to detect defections on the surface of a specific material (wood). First, we make sets of rules. Each defection type has a set of rules associated with it. The rules mapping functions are based on parameters and basic properties that define each defection type. Based on these parameters the system can detect all the defection types on the surface by calculating the confidence value after mapping the rules to the other defection properties on the surface. The example is illustrated through the following figure.
2.2. The finite state machine paradigm

A finite state machine is an approach used in order to represent several different types of sequences. Concerning the English language, finite state machines can represent strings of characters (not separated by spaces) that represent a word or variations of a word, or series of words (separated by spaces) that form a set of related phrases. [3].
As illustrated in the figure above, the finite state machine represents all the possible sequences of the English sentences. The output of this finite state machine is illustrated below.

**Results [3]**

see you later then
see you
I 'll see you next week then
have a nice day
I 'll see you tomorrow then
have a nice day
good night
take care
good luck
have a nice day
The finite state machine approach can be used in many different ways:

- **Generate random examples of sequences:**
  Take a path through the graph in figure 2 representing the finite state machine.

- **Recognize or parse sequences of elements:**
  A finite state machine can be written as a transducer that has an output string as well as input string on every edge. The transducer brackets the phrases. A **transducer** is a piece of software that maps one stream of symbols on to another stream of symbols. We say that the program transduces one stream of symbols into another.

- **Generate all possible sequences:**
  To do this you use a depth or breadth-first traversal of the whole graph.
• To generate sequences that meet some criteria, possibly semantic:
The most difficult use and also the most useful. For example, how do we carry out
the language function of saying goodbye in English? There are several phrases that
can be used and that are summarized in the finite state machine "farewell" above in
figure 2. [3].

2.3. The statistical approach

The statistical approach is based on the statistics data about the co-occurrence of
words that can be modeled easily. The statistics data can be extracted from the language
corpora and is used to resolve word sense ambiguities in real life applications. Take as
an example, the translation of the English word “in” to French. In this example, we
might expect that the translator always choose among the following:
{dans, en, a, au cours de, pendent}. The statistics data is the following equation:
\[ p(\text{dans}) + p(\text{en}) + p(\text{au cours de}) + p(\text{pendent}) = 1. \]  [4].
There are three aspects used when using the statistical approach.

1- Source of statistical knowledge:

There are two main sources of statistical knowledge, the local and global contexts. In
the local context, statistical data can be on the co-occurrence of syntactically related
words with each of their alternative senses. Parsers provide syntactic relations between
words (statistical data). However, robust syntactic parsers are not widely available.
In global context, statistical data can also be provided from words that are provided in
predefined vicinity around ambiguous words. The window of words surrounding each
ambiguous word sense can vary from 10 to 100 words. Predefined statistical data is
entered in the corpus.
Generally, each knowledge source complements the other. So the perfect natural
language processing system is the one that uses many knowledge sources since each
source provides data not captured by others.

2- Techniques used to acquire the required knowledge:

The best technique is to train the statistical model based on a corpus that is manually
tagged with the appropriate sense of each occurrence of an ambiguous word.
However, tagging the corpus manually is costly especially that the larger the corpus the
better is. A way to avoid the high cost of manual tagging is bootstrapping.
Bootstrapping is a method based on implementing the corpus incrementally.
Bootstrapping is achieved as follows:
We first build a small corpus manually, C1, and then train a model M1 using C1. C1 will
be used to disambiguate the rest of words in the corpus. All disambiguated occurrences
will merge with C1 to form C2 and so on.
Another technique is to use a bilingual corpus. A word E1 is aligned to different
words F1 and F2 in another language. F1 and F2 are tags for two different senses of E1.
Statistical data is obtained from the number of occurrences of such alignments.
A better technique is the bilingual lexicon and the monolingual corpus. The source word is mapped to the target word using the bilingual lexicon. Then a monolingual corpus is used to estimate the occurrence statistics in the second language.

3- Decision model for disambiguation:

A decision model is necessary to evaluate the statistics and resolve ambiguities based on these statistics. Many decision algorithms are used. One of these algorithms is the constraint algorithm that resolves ambiguities simultaneously as explained above. [5].

2.4. The Object Oriented Approach

The object oriented approach deals with building a base system (framework) that can be reused by any NLP application programmer by plugging other components in the base system. The system should run on a distributed and concurrent environment. Two core architectures can be used as a base for any NLP system especially text processing systems. There are two important natural language processing factors considered as the driving forces of the two architectures presented later in this section. These are the text presentation and the algorithms that process that text. These factors originate from two NLP architectural paradigms used to develop the base system. The two paradigms are:

- A processing paradigm in which the process is the driving force. In this approach, a process accepts the text and processes it.
- A text paradigm in which the text is the driving force. In this paradigm, a text is the driving force. A text is processed by taking a process and applying it to itself.

Both architectures use a text model.

![Figure 3: The text model][6]

The text model is a set of words organized hierarchically (Text, chapter, paragraph, sentence). A text model can contain text data views since a processing routine can’t process the whole model or hierarchy. Therefore, a text data view (TDV) can be used to
let the routine process the necessary text only. The changes made to the TDV propagate to the hierarchy used by other routines. The TDV is used to access only a particular subset of the text.

2.5. The processing paradigm

Also called **Pipeline and Dynamic Behavior**, the processing paradigm originates from the traditional way of procedural thinking. According to this approach, the text should be processed through several processing steps in order to obtain the desired result. This way of working can be seen as sending a text through a pipeline of processing steps. A pipeline is modeled using different pipeline processes that can be connected to each other according to a specific configuration. An example of a pipeline is illustrated in Figure 4.

![Figure 4: A pipeline example in NLP applications [6]](image)

Figure 4 illustrates a set of possible processing steps in a translation system. A text is submitted to the first processing step. When the step is finished the text is sent to the next processing step. When the last step exits, the result is returned to the caller of the pipeline. For a framework this model is not flexible enough. Often another pipeline setup will be needed when certain conditions are met. Therefore, in order to be able to change the pipeline, we introduce a pipeline monitor. When a text needs to be processed it is submitted to the pipeline through the pipeline monitor.
Figure 5: The Pipeline Monitor enables dynamic pipelines [6]

Figure 5 above shows the way the PM interacts with the actual pipeline. The pipeline monitor first selects a first step in the pipeline and sends the text to this step. The particular processing step accepts the text, handles it and then returns the resulting text back to the pipeline monitor. The pipeline monitor then selects the following processing step, submits the text, etc. When certain conditions are met the pipeline monitor can change the pipeline by inserting processing steps or taking alternative processing steps as can be seen on the figure.

A typical example of the processing paradigm is the machine translation example shown in figure 6 below. In this example, the text is submitted to a text pipeline monitor. The text pipeline monitor submits the text to an ML process that does a morphological lookup on the text. The translation process (TL) decomposes the text in sentences and sends it to the Sentence pipeline monitor. The Sentence Pipeline Monitor then does the actual translation of each sentence by calling the Parsing (PA) and transfer (TR) processes. The parsing process creates representations of the first language sentences. The transfer process transfers each word in the representation to its translation in terms of the second language. Finally, the translated sentence is passed to the caller of the pipeline, which is the translation process (TL). After the text is translated, a new generation process (GE) starts. The generation process processes the representation returned by the translation process and generates the translated sentence in the second language.
2.6. The Text approach

The text paradigm originates from a more object-oriented background. Instead of having an algorithm that processes a text, the text will process itself. This means that next to the data representation, the text encapsulates information on how to process itself. The text model we use is the same as presented in figure 3. The processing components are encapsulated in text visitors. Text visitors are a set of methods that the text uses to process itself. In the text paradigm, a supervising unit, called the processor selects several text visitors in some order, hands them to the text and asks the text to process itself using these text visitors. Generally, the processor knows what needs to be done, step-by-step. The text visitors are the steps that know how to do it, as long as the processor sends them to the text in the right order.

Figure 7 clarifies this approach with an example. The example is about the implementation of the translation system using the same text data view used in figure 3. First of all, the processor chooses a morphological lookup (ML) component and asks the text to accept the component. After acceptance, the text starts processing itself by calling the `handleText()` method. `handleText()` takes the title of the text, which can be retrieved from the root text component of the text structure, and starts looking up the words from the title. When `handleText()` finishes looking up the words, `handleText()` starts iterating over the children of the Text object (chapters) by sending the ML component to these subcomponents. The chapter objects also accept, but this time `handleChapter()` is called. The words in the title of the chapter are looked up, and then `handleChapter()` iterates again over the paragraphs in the text. This continues until the leaves in the text structure are looked up. The ML component finishes and the processor selects the following component, such as a parsing component (PA). The parser is sent over the text structure in the same way, parsing the text title, chapter title and so on.
The two approaches can be seen as opposite point of views. The process point of view deals with sending a text through several processes to obtain a result while the text point of view deals with sending several processes through a text. Although the two models can be seen as opposites, they have a lot in common too. They share the same text model. They both offer the possibility to take run-time processing decisions based on text characteristics. This is obtained by using pipeline monitors in the process point of view and processors in the text point of view.

3. Main natural language processing tools

Natural language processing systems are based on many tools that can be used to provide a better performance in understanding human beings language. Taggers, parsers and lexicons are the backbone of NLP systems. In this paper, we are going to focus mainly syntactic parsers and lexicons.

3.1. Syntactic parsers (Link Grammar)

Syntactic parsers are programs that receive inputs from interactive online commands and break them into parts that can be managed by other programs. A very good and robust parser is the Link Grammar parser. Given a sentence, the Link Grammar assigns to it a syntactic structure, which consists of a set of labeled links connecting pairs of words. “The parser is robust; it is able to skip over portions of the sentence that it cannot understand, and assign some structure to the rest of the sentence. It is able to handle unknown vocabulary, and make intelligent guesses from context and spelling about the syntactic categories of unknown words. It has knowledge of capitalization, numerical expressions, and a variety of punctuation symbols.” [7]. The Link Grammar parser can be integrated in NLP applications through its API. The Link Grammar API defines five basic structures. In order to parse a sentence and extract information from it, the user
creates and manipulates these types using different function calls. Following is an overview of these five data structures:

- **Dictionary**: A Dictionary defines the set of word definitions that defines the grammar. A user creates a Dictionary and passes it to the various parsing routines.
- **Sentence**: The input string, tokenized and interpreted according to a specific Dictionary.
- **Parse Options**: Parse_Options specify the different parameters that are used to parse sentences. Examples of the kinds of things that are controlled by Parse_Options include maximum parsing time.
- **Linkage**: This is the API's representation of a parse. A Linkage can be constructed after a sentence has been parsed, and can be thought of as a Sentence together with a collection of links.
- **Postprocessor**: A PostProcessor is associated with each Dictionary, and automatically applied after parsing each Sentence constructed using that dictionary. Individual linkages can be post-processed with different sets of context-sensitive post-processing rules. The API enables this by letting the user open up a set of rules and passes it around as a PostProcessor.

### 3.2. Link Grammar Example [7]

The following example is about a program that opens up a dictionary and then parses two sentences, graphically displaying a linkage for each.

```c
#include "link-includes.h"

int main() {

    Dictionary    dict;
    Parse_Options opts;
    Sentence      sent;
    Linkage       linkage;
    char *        diagram;
    int           i, num_linkages;
    char *        input_string[] = {
        "Grammar is useless because there is nothing to say -- Gertrude Stein.",
        "Computers are useless; they can only give you answers -- Pablo Picasso."};

    opts  = parse_options_create();
    dict  = dictionary_create("4.0.dict", "4.0.knowledge", NULL, "4.0.affix");

    for (i=0; i<2; ++i) {
        sent = sentence_create(input_string[i], dict);
        num_linkages = sentence_parse(sent, opts);
        if (num_linkages > 0) {
            linkage = linkage_create(0, sent, opts);
        }
    }

    // Further code...

```
The statements:

```c
opts  = parse_options_create();
dict  = dictionary_create("4.0.dict", "4.0.knowledge", NULL, "4.0.affix");
```

create Parse_Options and a Dictionary to be used in processing sentences. In order to create a dictionary, the program looks in the current directory for the files 4.0.dict and 4.0.knowledge.

```c
sent = sentence_create(input_string[i], dict);
```

takes the sentence as input and uses the Dictionary that was created earlier to tokenize and create word definitions such as (noun, verb, object...etc). The statement

```c
num_linkages = sentence_parse(sent, opts);
```

passes the sentence, along with the Parse_Options, to the function sentence_parse, which returns the number of all possible linkages. The following statements

```c
linkage = linkage_create(0, sent, opts);
printf("%s\n", diagram = linkage_print_diagram(linkage));
string_delete(diagram);
linkage_delete(linkage);
```

extract the first linkage indexed by 0 in the list, prints the linkage diagram and then deletes linkage and the string allocated for the diagram. After each of the input strings is processed, and for memory management purposes, the program deletes the Dictionary and Parse_Options with the statements:

```c
dictionary_delete(dict);
parse_options_delete(opts);
```

The program output is illustrated in the following figure.
Two main Link Grammar API functions are:

```c
char ** linkage_get_words(Linkage linkage);
char * linkage_get_word(Linkage linkage, int w);
```

- `linkage_get_words`: Takes a linkage as an argument and returns the array of word spellings of the current linkage. Ex. Computers, are, etc.
- `linkage_get_word`: Returns one linkage word at a time based on the index `w`. These words can be used by other NLP applications components such as lexicons.

### 3.3. Parsers evaluation methods

Parsers play a very crucial role in Natural Language processing applications performance. Therefore, smart developers should always evaluate and test different kinds of parsers before integrating them into their applications. Parsers are either dependency based or constituency-based systems. Constituency based parsers produce a hierarchically organized constituent structure while dependency based parsers produce linkages between words of sentences [sri]. Parsers are potentially important in every natural language processing system. The parser quality mostly reflects the system performance. Parsers and choosing the right one depends on the evaluation method chosen. According to [sri], three evaluation methods are available.

- Intrinsic
- Extrinsic
- Comparative

Intrinsic evaluation measures the performance of a parser in the context of the framework in which it is developed [sri]. This approach can be used for both grammar based and statistical parsers. In grammar-based parsers, intrinsic evaluation checks the grammar vulnerabilities. In statistical parsers, the underlying statistical model performance is measured. Intrinsic evaluation is divided into two approaches, test suite based and corpus based evaluations. Corpus based evaluation is subdivided to annotated and unannotated corpus based evaluation methods. Intrinsic evaluation also uses the test suite based approach. It gives good direction on the improvement of statistical parsers. This approach is not suitable for unrestricted text data.
Intrinsic method also uses two additional approaches: unannotated corpus-based and annotated corpus-based. The first approach uses unrestricted text data as corpora in order to evaluate parsers. The sentences in corpora are not annotated with any linguistic information. The annotated corpus-based approach uses unrestricted text data formed of sentences that are annotated with information such as part of speech tags, subject-verb-object triples... etc [10]. This methodology is very efficient in evaluating the parser tagging as well as parsing accuracies. One can measure the exact match between the structure output by the parser and that of the corpus annotation.

Extrinsic evaluation: Mainly measures the parser adequacy. In other words, it is more suitable for developers who want to integrate a parser into their systems. By using the extrinsic evaluation, the developer can see how the parser contributes to the all system performance. This method provides an indirect parser comparison too.

Comparative evaluation: Comparative evaluation is appropriate for comparing parsers that use different grammar or statistical models. This method uses the PARSEVAL metric that proved many limitations such as crossing brackets penalization to mis-attachements.

In the model proposed by Bangalore, Sarkar, Doran and Hockey, a parser is viewed in multiple dimensions where each dimension represents a relation R. The performance of the parser is measured in terms of precision (P), recall (R) and F-measure. A summary of the evaluation method is illustrated in figure 1.

- Let \( x R y \) represent that \( x \) is in a relation \( R \) with respect to \( y \).
- Let \( S_{\text{gold}} \) be the relation, \( R \), expressed in the key (annotated corpus).
- Let \( S_{\text{out}} \) be the relation, \( R \), expressed in the output of the parser.
- Recall = \( \frac{|S_{\text{gold}} \cap S_{\text{out}}|}{|S_{\text{gold}}|} \)
- Precision = \( \frac{|S_{\text{gold}} \cap S_{\text{out}}|}{|S_{\text{out}}|} \)
- F-Measure = \( \frac{(\beta^2 + 1) \cdot P \cdot R}{\beta^2 \cdot P + R} \) where \( \beta \) is the relative importance of Recall and Precision.

**Figure 9: The relational model for parser evaluation summary[8]**

Other evaluation method, that we would consider, are summarized below [8]:
- Listing Constructions: Consists of simply listing covered constructions that may not be covered by the evaluated parser.
- Coverage (Unannotated corpus): This includes the calculation of a percentage of examples from a given corpus, which are assigned one or more global analyses by a parser.
- Average parse base (Unannotated corpus): The evaluation is based on computing the geometric mean of the analyses divided by the number of input tokens in each parsed sentence. This produces the average number of parses for a sentence of n words.
- Tagging Accuracy: (Annotated corpus): The evaluation is done by calculating the accuracy with which a parser assigns the correct lexical syntactic category to a word running in text. Precision / recall are the best measure.
- Tree similarity measure (annotated corpus): Evaluation is performed through the calculation of the ratio of rules correctly deployed in a derivation to all rules in that derivation computed from an annotated corpus.

3.4. Lexicons (WordNet)

WordNet is an online lexical database that contains English nouns, verbs, and adjectives organized into synonym sets (synsets) such as {board, plank}. These synonym sets do not explain what the concepts are; they only explain that there is a concept that exists. According to [Miller], WordNet is organized by semantic relations. Since a semantic relation is a relation between meanings, and since meanings can be represented by synsets, it is natural to think of semantic relations as pointers between synsets. There are four main divisions in WordNet: Adverbs, nouns, verbs and adjectives. WordNet has many semantic relations that the developer should focus on when developing natural language processing systems. Each WordNet division has its own relations that relate the division corresponding synsets. Following are the main WordNet relations.

**Synonymy**
Two expressions are synonymous if the substitution of one for the other never changes the truth value of a sentence in which the substitution is made. For example, “the substitution of *plank* for *board* will seldom alter truth values in carpentry contexts.” [9].

**Antonymy**
The antonym of a word x is sometimes not-x, but not always. Ex. Rich and poor are antonyms. But not rich doesn’t mean poor. “Antonymy provides a central organizing principle for the adjectives and adverbs in WordNet, and the complications that arise from the fact that antonymy is a semantic relation between words are better discussed in that context.” [Miller].

**Hyponymy/ Hypernymy (IS-A relation)**
Also named subset/superset relation. For example, a mapple is a hyponym of a tree. A hyponym always inherits the features of its hypernym.

**Meronymy/Holonomy**
“A concept represented by the synset \{x, \ldots\} is a meronym of a concept represented by the synset \{y, \ldots\} if native speakers of English accept sentences constructed from such frames as \textit{A y has an x (as a part)} or \textit{An x is a part of y}. The meronymic relation is transitive (with qualifications) and asymmetrical (Cruse, 1986), and can be used to construct a part hierarchy” [Miller]. The following figure illustrates how WordNet semantic relations are represented.

![Figure 10: Representation of three semantic relations among a variety of lexical Concepts [9].](image)

**Entailment**

Entailment is a semantic relation that relates verb synsets. “A proposition \(P\) entails a proposition \(Q\) if and only if there is no conceivable state of affairs that could make \(P\) true and \(Q\) false.” [9]. An example of entailment is the following: \textit{snore} lexically entails \textit{sleep} because the sentence \textit{He is snoring} entails \textit{He is sleeping}; the second sentence necessarily holds if the first one does.

4. **An NLP sample application**

The main goal of the sample application is to get all the possible meanings of every verb in a certain sentence.

4.1. **General Requirements**

- The application takes a sentence as input through an interactive interface.
- The application should locate the main verb in the sentence.
- The verb is provided to WordNet lexicon for processing.
- Wordnet takes the verb and outputs all the synonyms of that verb.
- The application can be extended further by applying some AI algorithms to the verb meanings output by WordNet.
- The algorithm chooses the right semantic meaning of the verb in the sentence.

4.2 The application Design

![Diagram: NLP sample application design]

Figure 11: The NLP sample application design

As illustrated in the figure above, the user will input a sentence in the form of a string. The Link Grammar parser will process the sentence and outputs a representation or a linkage containing word definitions. In fact the parser will divide the input sentence into different words (nouns, verbs, objects, etc). The different words will be stored as an XML document. The application will take the XML document and read the verb only. The verb is input to WordNet lexical database. WordNet will process the verb and output all the possible verb meanings.

4.3 The Output

Both the parser and WordNet parts are implemented separately. However, they need to be integrated. The integration is accomplished using the java native interface (JNI). The application output is illustrated in the following figures.
5. Conclusion

Natural language processing has been a very important field especially after the recent events. Scientists should concentrate more on improving NLP applications because computers are always inferior to humans because of they are unable to understand sentences from their contexts. But we never know, one day computers will replace humans.
References

[1] www.microsoft.com/nlp/research
Department of Computer Science K.U.Leuven
http://www.link.cs.cmu.edu/link/api/index.html
[8] Introduction to WordNet: An On-line Lexical Database George A. Miller,
Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine Miller
(Revised August 1993).
[9] Grammar & Parser Evaluation in the XTAG Project, Srinivas, Bangalore,
Anoop Sarkar, Christine Doran, Beth Ann Hockey.