UCSG Shallow Parser:
A Hybrid Architecture for a Wide Coverage Natural Language Parsing System

A Thesis Submitted

for the Degree of

Doctor of Philosophy

in

Computer Science

by

Guntur Bharadwaja Kumar

Department of Computer and Information Sciences

School of Mathematics and Computer/Information Sciences

University of Hyderabad

Hyderabad - 500046, India

APRIL 2007

Declaration

I, Guntur Bharadwaja Kumar, hereby declare that the work presented in this thesis has been carried out by me under the supervision of Prof. Kavi Narayana Murthy, Department of Computer and Information Sciences, University of Hyderabad, Hyderabad, India, as per the Ph.D ordinances of the University. I declare, to the best of my knowledge, that no part of this thesis has been submitted for the award of a research degree of any other University.

Guntur Bharadwaja Kumar,

Registration No: 01MCPC05.

Certificate

This is to certify that the thesis work entitled "**UCSG Shallow Parser: A Hybrid Architecture for a Wide Coverage Natural Language Parsing System**" being submitted to the University of Hyderabad by **G. Bharadwaja Kumar** (Reg. No. 01MCPC05), for the award of the degree of Doctor of Philosophy in Computer Science, is a record of *bona fide* work carried out by him under my supervision.

The matter embodied in this report has not been submitted to any other University or Institution for the award of any degree or diploma.

Prof. Kavi Narayana Murthy
Supervisor,
Department of CIS,
University of Hyderabad,
Hyderabad–500046

Prof. Arun Agarwal
Head,
Department of CIS,
University of Hyderabad,
Hyderabad–500046

Prof. T. Amaranath
Dean,
School of MCIS,
University of Hyderabad,
Hyderabad–500046

Dedicated to

My Uncle

Sri. P. V. Anjaneya Charyulu

Abstract

Natural Language Processing (NLP) is an area which is concerned with the computational aspects of human languages. The aim is to build computational systems for generation, analysis, and understanding of natural languages. Within this scenario, *Syntax* deals with the characterization of structure of natural language sentences and in turn plays an important role in several levels of natural language processing. Natural language sentences exhibit an extremely rich and varied structure and are often ambiguous. Hence, syntactic analysis or parsing has remained a challenging task even after several decades of research. The performance of many natural language processing applications such as Information Extraction, Question-Answering and Machine Translation systems is limited today by the performance of syntactic parsers they use.

Although a lot of work has gone into developing full syntactic parsers which identify both structural and thematic relations, it has not been possible to achieve high performance on unrestricted texts. Given this scenario, there has been an increased interest in wide coverage and robust but partial or shallow parsing systems in the last decade or so. Shallow parsing is the task of recovering only a limited amount of syntactic information from natural language sentences. It has been found that even partial or shallow parsing has many important uses and applications.

There are broadly two approaches for the development of parsers - the linguistic approach which depends upon hand-crafted grammars, and the machine learning approach where parsers are learned automatically from a labeled training corpus. Developing hand-crafted grammar rules is a very slow, tedious and difficult task, requiring substantial knowledge and skill on the part of the linguist. Automatic learning of grammars requires, on the other hand, a large, representative, parsed training corpus, which is rarely available. Perhaps only a good combination of linguistic and statistical approaches can give us the best results with minimal effort.

In this thesis, we have proposed a methodology for building wide coverage shallow parsers by a judicious combination of linguistic and statistical techniques without need for a training corpus to start with. We have proposed an architecture, called UCSG Shallow Parsing Architecture, which uses a judicious combination of linguistic and statistical approaches for building wide coverage shallow parsing systems. It uses the philosophy of UCSG (Universal Clause Structure Grammar) syntax for parsing proposed by Kavi Narayana Murthy at University of Hyderabad, hence the name 'UCSG' shallow parsing architecture. The input to the parsing system is one sentence, either plain or POS tagged. Output is an ordered set of parses. The aim is to produce all possible parses in ranked order hoping to get the best parse to the top. In this work, by parse we mean a sequence of chunks. Chunks are sequences of words. A chunk or a "word group" as we prefer to call it in UCSG, is "a structural unit, a non-overlapping and non-recursive sequence of words, that can as a whole, play a role in some predication".

In the UCSG architecture, a finite state grammar is designed to accept *all* valid word groups but not necessarily *only* those word groups that are appropriate in context for a given sentence. From literature we have seen that simultaneous satisfaction of both the requirements has proved very difficult in practice. A separate statistical component, encoded in HMMs (Hidden Markov Model), has been used to rate and rank the word groups so produced. Note that we are not pruning, we are only rating and ranking the word groups produced. Then we have used a best first search strategy to produce parse outputs hopefully in best first order, without compromising on the ability to produce all possible parses. We have also proposed a bootstrapping strategy for improving HMM parameters and hence the performance of the parser as a whole.

In this work, we have built a dictionary that includes words, POS tags, frequency of occurrence for each tag for each word from the British National Corpus (BNC). We have developed a manually parsed corpus of 4000 sentences according to UCSG syntax by taking sentences from a wide variety of sources. We have shown that finite state machines are sufficient to produce all valid word groups for a given sentence. We have evaluated FSM module in terms of the number correct chunks it can recognize i.e recall. We have achieved a high recall of 99.5%

on manually parsed corpus, 95.06% on CoNLL test data and 88.02% on Susanne corpus. We have successfully built HMMs by using only POS tagged BNC sentences and used them for rating and ranking word groups. We have evaluated the performance of the HMM modules in terms of mean rank score i.e. mean of the distribution of ranks of the correct chunks in the manually parsed corpus. We have obtained good mean rank scores i.e. 2.26 for plain sentences and 1.57 for POS tagged sentences on a test data of 4000 sentences.

We have proposed a best first search algorithm to select the best chunk sequence as a parse for a given sentence. All possible parses are produced but in a best first order. When we restrict best first module to give best five parses and time limit to 3 epoch seconds, we have obtained 45.52% correct parses within the top 5 for plain sentences and 68.02% of correct parses within the top 5 for POS tagged sentences. The percentage of correct chunks in the top parse is 78.70 for plain sentences and 78.42 for POS tagged sentences. We have also used a modified beam search method but here we may loose some of correct parses because of pruning although substantial speed up can be achieved. The number of correct parses in the top position for plain sentences has also increased from 28.25% to 31.55%. Number of sentences that can be parsed within the stipulated time increased from 54.67% to 100%.

We have proved our idea of bootstrapping to improve HMMs parameters as well as the performance of the whole parser. We have done bootstrapping in three ways: by taking HMM top ranked chunks, chunks from top parse given by third module and both combined. We have found that bootstrapping from top of parse of best first search module gives best results. We are able to improve the mean rank score to 2.21 from 2.26 in the first iteration and to 2.16 in the second iteration. The performance of the parser also improved in terms of pushing correct parses to the top. Before bootstrapping, there were 28.25% correct parses in the top position for plain sentences and this improved to 30.25%. The percentage of correct chunks in the top parse improved from 78.70% to 83.92%. For tagged sentences, the percentage of correct chunks in top parse improved from 78.42% to 88.26%. The percentage of correct parses in top position improved from 44.35% to 54.82%.

In this work, we have described our experiments on English language using the British National Corpus and the results are very encouraging. We believe that the methodology is applicable to Indian languages as well. It should be possible to develop large scale computational grammars and parsers for Indian languages too. Raw text corpora as well as morphological analyzers are available in many languages and POS-tagged corpora can be developed easily. Developing Finite State Grammars should also be relatively easy. We have done some preliminary experiments including corpus analysis to understand the nature of Indian languages. We have also developed a system for language identification from text samples of Indian languages. The results are comparable to the best published results in the world.

Acknowledgements

I am very grateful to my advisor, Prof. Kavi Narayana Murthy, for his unfailing support throughout my Ph.D. He has great insight and ability to make complex ideas easy and research fascinating. As my advisor, he was incredibly patient with many of my failures and his trust in my abilities helped me to overcome many difficulties. It is a great honour for me to work under such an eminent and versatile personality.

My sincerest thanks to my Doctoral Committee members Prof. Hrushikesha Mohanthy and Dr. S. Bapi Raju. They were always encouraging and helpful in my research.

My sincerest thanks to Dr. S. Durga Bhavani for many insightful conversations, for her constant help and encouragement throughout my Ph.D.

My sincerest thanks to Prof. Arun K. Pujari for his valuable suggestions and encouragement throughout my Ph.D.

My sincerest thanks to the Head of the Department Prof. Arun Agarwal and all the other members of the faculty of our department for their encouragement.

I am indebted to Prof. V. S. S. Sastry whose able guidance divulged the knowledge and skills that are dormant inside me.

My sincerest thanks to Prof. K. P. N. Murthy whose inspiration made me a good researcher.

My sincerest thanks to Prof. C. Raghavendra Rao for his valuable suggestions.

My sincerest thanks to Prof. O.B.V. Ramaniah for his valuable suggestions and support.

My sincerest thanks to the U.P.E. committee for providing me fellowship when I was in great need.

My sincerest thanks to my well wishers Dr. Vamanan Satheesh, Dr. Anitha Semwal, Dr. M. Sree Ramana and Dr. L.N.S. Prakash Goteti.

My special thanks goes to my friend Ahmed who helped me whenever I was in great need.

I would like to thank my dearest friends B.V.N Phani Kumar, P.V.S. Chandrasekhar, N.K.M.Naga Srinivas, R. Phani Murali Krishna, M.Venkat Reddy, P.S.V.S. Sai Prasad, M. NagaMani, M. Trivikram Rao, S. Nageswara Rao, S. Mahipal, T. Sivaram, C. T. Mohan, Satyajit Acharya, Hla Hla Htay, M. Ravindranath, K. Nagesh, Bira Chandra Singh, B. Shankara Rao, M.S.R. Varma, Subrath Kumar Dash.

Finally, I am forever indebted to my parents and family members for their understanding, endless patience and encouragement when it was most required.

Guntur Bharadwaja Kumar

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Language is the fundamental means of communication for human beings. With the revolution of electronic and information media in global communication, natural language texts in electronic form have become a principle medium for communication. We have billions of pages of textual data online today. This necessitates an easy way to access relevant, useful and well presented information. Since the electronic documents and texts are natural language constructs, language specific knowledge is indispensable for the processing of these documents. This brings the necessity to deal with computational aspects of human languages.

Natural Language Processing (NLP) is an area which is concerned with the computational aspects of the human language. The goal of the NLP is to analyze and understand natural languages used by humans and to encode linguistic knowledge into rules or other forms of representation. Natural language understanding is sometimes referred to as an AI-complete problem, because natural language understanding requires extensive knowledge about the outside world and the ability to manipulate such knowledge. Even though machines have proven capable of doing complex operations such as inverting large matrices, they still fail in natural language learning tasks.

The main difficulty in natural language processing tasks is perhaps its ambiguity. Ambiguity in natural language pervades virtually all aspects of language analysis. Sentence analysis in particular exhibits a large number of ambiguities that demand adequate resolution before the sentence can be understood. Most of the language processing applications like Information Retrieval (IR), Information Extraction (IE), Question-Answering systems, Speech Recognition and Synthesis, Text Summarization and Machine Translation (MT) are affected by the highly ambiguous nature of natural language.

Ambiguities in sentence analysis are generally categorized into two types: lexical and structural ambiguities. Structural ambiguities can be further divided into prepositional phrase attachment, referential ambiguity, reduced relative clauses etc. A classic example of prepositional phrase attachment in English is shown below:

Put the block in the box on the table.

There are two possible analyses for the above example:

Put the block [in the box on the table].

Put [the block in the box] on the table.

If we add another prepositional phrase (say, "in the kitchen"), we obtain five analyses, one more (say, "through the window"), we obtain fourteen and so on. More specifically, the degree of ambiguity follows a combinatoric series called Catalan numbers, in which the n^{th} number is given by [33]

$$C_n = 2n_{C_n}\frac{1}{n+1} \tag{1.1}$$

Lexical ambiguity is also the source of much non-determinism in parsing. Lexical ambiguity arises when a lexical item has alternate meanings and different POS tags.

Syntax deals with the characterization of structure of natural language sentences. It concerns how different words are combined into phrases, phrases into clauses, which, in turn, are combined into sentences. Parsing is the analysis of syntactic structures in a sentence according to a formal grammar. Parsing systems perform syntactic analysis of the natural language text using computational grammars specified in some suitable grammar formalism.

A variety of grammar formalisms have been proposed for parsing: Case Grammars [43], Government Binding theory [25, 130], Lexical Functional Grammar [66, 65, 130], Definite Clause Grammars [1], Functional Unification Grammar (FUG) [67], Generalized Phrase Structure Grammar[44], Tree Adjoining Grammar [51, 62, 132], Universal Clause Structure Grammar [95, 94, 93], Dependency Grammars [57], Categorical Unification Grammar (CUG) [38] , Head Driven Phase Structure Grammar (HPSG) [111], Link Grammar [50, 134], PATR, Probabilistic Context Free Grammar [24, 68], Minimalist Approach [148, 100], Paninian Grammar [11] etc.

One important issue in developing a grammar formalism which defines wide coverage grammars is that grammar development is extremely laborious. Another important issue in grammar engineering is the reusability of grammars. The more a grammar is committed to a certain processing model, the less are the chances that it can be adapted to other processing models or new application areas. This constitutes a serious bottleneck in the development of language technology products. Many of the grammar formalisms discussed above suffer from these disadvantages. Some of the linguistic grammar formalisms have not considered computational viability as an important factor.

Grammar formalism based full parsing systems try to give complete analysis of sentence i.e. both basic structural descriptions and thematic roles. Even though grammar formalism based full parsing systems are generally able to correctly model all grammatical functions and thematic roles, they gives relatively low accuracy because of 1) exponential solution space 2) dependence on semantic and pragmatic knowledge 3) long-distance dependencies 4) ambiguity in grammars and 5) error propagation. Often there will be many parses for a given sentence and selecting best parse automatically is difficult. It is often quite difficult to determine the suitability of a grammar formalism for a particular task.

Over the years, many NLP applications have adapted different grammar formalisms. Parsers that adopted purely linguistic grammar formalisms have not been very successful in terms of wide coverage and high accuracy. For example, an XTAG parser is a parser based on Tree Adjoining Grammar formalism that includes wide coverage syntactic grammar of English. The range of syntactic phenomena that can be handled is large and includes auxiliaries (including inversion), copula, raising and small clause constructions, topicalization, relative clauses, infinitives, gerunds, passives, adjuncts, it-clefts, wh-clefts, PRO constructions, noun-noun modifications, extraposition, determiner phrases, genitives, negation, noun-verb contractions and imperatives etc. When this XTAG parser was tested on Wall Street Journal, IBM manual, and ATIS corpora, the performance was very low [39]. See table 1.1.

Table 1.1: Performance of XTAG on Various Corpora

Corpus	No of sentences	% Parsed	Av. No. of parses per sent
WSJ	6364	39.09%	7.53
IBM Manual	1611	75.42%	6.14
ATIS	649	74.42%	6.0

Recently, statistical and machine learning algorithms have taken a lead over complex linguistic grammar formalisms in solving the problems of parsing. There has been a great progress in natural language processing, through the use of statistical methods trained on large parsed corpora. Statistical parsing approaches are able to tackle the ambiguity problem by assigning a probability to each parse tree, thereby ranking competing trees in order of plausibility. Compared to conventional grammars, probabilistic grammars have preformed better in parsing.

Table 1.2: Labelled Precision(P), Labelled Recall(R) of Recent Statistical Full parsing Systems < 40 words

Parser	LP (%)	LR (%)
Collins (1999)	88.7	88.5
Charniak (2000)	90.1	90.1

Even though the probabilistic systems are capable of working with naturally occurring language exceptions, we face some difficulties in terms of long sentences, widely varying vocabularies and unexpected constructions. One more drawback of statistically based parser is that it requires large quantities of annotated data for training. Creating tree banks or parsed corpora is a Herculean task, so there will not be many to choose from. Thus the variety of parse types generated by the systems may also be limited. Typically, the training data must consist of real sentences annotated with structural information of the kind the parser will eventually generate. Unfortunately, annotating these sentences can require a huge amount of work by language experts, comparable to that required to develop a rule-based grammar.

One more problem with full parsers is that they try to give more information than required for some applications. For example, named entity recognition itself is very much useful result for many NLP applications such as information retrieval. Also, tree based outputs are not useful for many of the NLP applications where only phrases are required for the task. There has been an increased interest in wide coverage and robust but partial or shallow parsing systems in the last decade or so. Shallow (or partial) parsing is the task of recovering only a limited amount of syntactic information from natural language sentences.

Shallow parsing does not attempt to resolve all semantically significant ambiguities. Most of the times it is restricted to identifying phrases in sentences. In CoNLL chunking task [124], chunking was defined as *the task of dividing the text into syntactically non-overlapping phrases.* Here by 'non-overlapping' we mean, one word can become a member of only one chunk.

Shallow parsing has proved to be more practical because of higher processing speed, lower costs in the design and maintenance of grammars with a corresponding reduction of the output information. A good trade-off between expressiveness and efficiency in shallow parsers is achievable.

Chunking provide an intermediate step to full parsing. Although identifying whole parse trees can provide deeper analyses of the sentences than merely identifying chunks, full parsing is a much harder problem. Given today's technology, machines can identify chunks much more accurately than they can identify parse trees.

Table 1.3: Performance of the recent shallow parsing systems

Parsing System	Precision	Recall	F-measure
Zhang , Damerau , David Johnson	94.28	94.07	94.17
Kudoh and Matsumoto	93.45	93.51	93.48
Van Halteren	93.13	93.51	93.32

Literature survey and detailed analysis show that:

- Even shallow or partial parsing of natural languages is quite challenging.
- Testing and evaluation of shallow parsers has been carried out only on limited amount of data (say, 2000 sentences from WSJ corpus) in most cases. Performance on large scale real life data is not clear.
- Testing and evaluation of parsers is a difficult task. Parsing accuracy of trained parsers is known to depend significantly on stylistic similarities between training corpus and test data. For example, Chris Huyck's plink parser [152] was trained on Wall Street Journal portion of the Penn Tree Bank (PTB) and when it was tested on Penn Treebank and Susanne corpus, there was a significant variation in parser performance. Daniel Gildea [46] studied variation of parser performance on different corpora and observed the same effect.
- Some of the most accurate parsers namely Collins parser, Charniak parser use lexical co-occurrence statistics in the parsing model. Daniel Gildea [46], in his paper states that "lexical co-occurrence probabilities seem to be of no benefit when attempting to generalize to a new corpus".
- High performance has been achieved only under restricted conditions. For example, in CoNLL 2000 chunking task[124] prepositions were not fully disambiguated, prepositional phrases not built and no attempt made to resolve ambiguities relating to attachment of prepositional phrases.
- In the literature, the performance of the shallow parsers is measured in terms of individual chunk types produced rather than the correct chunk sequence or parse.
- A parser also needs to have good generalization capacity for the other domains. Current systems have not been shown to be good at this.

- Most systems have used either a linguistic approach or a machine learning approach. There is a lot of scope for exploring combinations of linguistic and machine learning approaches in syntactic parsing.

- Given the richness of syntactic structure, large amounts of high quality parsed corpora are required for statistical approaches. The largest training corpora available for English are hardly a few hundred thousand sentences. In many languages of the world, hardly any parsed corpora are available. Further, training corpora must be suitable for a given grammar or grammar formalism. There are strong corpus effects.

- While labelled training data is difficult to build, large scale unlabelled training data (that is, plain or POS tagged text corpus) is readily available or can be easily developed. The challenge is to exploit this for developing wide coverage grammars and parsing systems.

- While several grammars and parsing systems exist for English and other major languages of the world, Indian languages are lagging far behind. There are hardly any substantial computational grammars for any of the Indian languages. Parsed corpora are also not available and hence machine learning approaches cannot be applied right away.

1.2 Objectives of Study

1.2.1 Broad Objective

To develop a methodology for building high performance wide coverage computational grammars and parsers by combining linguistic and statistical approaches in a judiciously designed hybrid architecture without need for large scale parsed training corpora to start with.

1.2.2 Specific Objectives

1. To propose a hybrid architecture for building shallow parsers which is suitable for both fixed and free word order languages.

2. To explore the advantages of judicious combination of linguistic and statistical techniques in developing wide coverage, robust shallow parsers.

3. To explore the effect of bootstrapping in developing parsers without need for large representative parsed training corpus to start with.

4. To explore the trade-off between Precision and Recall for identifying chunks. Ideally, we must obtain 'all' and 'only' valid combinations but satisfying both these requirements at the same time is very difficult in practice. To explore ways of separating the generalization and specialization requirements.

5. To explore grammars for very high Recall (all valid chunks to be recognized, possibly many more as well) without regard to Precision.

6. To explore techniques to rate and rank the chunks already recognized so as to push good chunks towards the top.

7. To explore techniques to select optimal chunk sequence so as to get the best parses on top without compromising on the ability to produce all possible parse outputs.

1.3 Methodology

UCSG full parsing framework is being developed at University of Hyderabad, India, since early nineties [93, 94]. UCSG framework is highly modular and amenable for hybrid extensions. However, UCSG had remained so far a purely knowledge based approach - there was no statistical component. No wide coverage grammars or parsers had so far been built using UCSG.

In this thesis, we propose an architecture, called UCSG Shallow Parsing Architecture, which uses a judicious combination of linguistic and statistical approaches for building wide coverage shallow parsing systems [75, 97]. See Figure 4.1 in chapter 4. The focus is only on recognizing word groups (also known as chunks) and to identify appropriate sequences of word groups to describe the structure of given sentences - clause structure analysis and functional structure analysis (including assignment of thematic roles such as subject and object) are beyond the scope of the current work.

The current thesis shows that Finite State Grammars with very high Recall can be built with relative ease. We then add a statistical component in the form of HMMs for rating and ranking the chunks produced by the finite state module. We finally describes how best first search can be used to produce appropriate chunk sequences or parses for a given sentence. This thesis also shows how bootstrapping technique can be used to improve the performance of the parser. A wide coverage shallow parsing system has been developed for English and its performance evaluated.

The input to the parsing system is one sentence, either plain or POS tagged. Output is an ordered set of parses. The aim is to produce all possible parses in ranked order hoping to get the best parse to the top. By parse we mean a sequence of chunks in this work. Chunks are sequences of words.

A chunk or a "word group" as we prefer to call it in UCSG, is "a structural unit, a non-overlapping and non-recursive sequence of words, that can as a whole, play a role in some predication [93]".

Note that word groups do not include clauses (relative clauses, for example) or whole sentences. Every word group has a head which defines the type of the group. Word groups can be classified into verb groups, noun groups, adjective groups and so on based on the essence of the meaning as indicated by the *head* of the word group. Thus word groups are similar to *chunks* [91, 124]. Our word groups are also very similar to the *phrases* defined in the work of Beata Megyesi [85]. It may be noted that the terms *chunk* and *phrase* have been used in substantially different connotations elsewhere in literature. For example, prepositions are treated as chunks in their own right in many chunking systems. The word groups we produce in UCSG are hopefully closer to ideal, semantically oriented units of full parsing, as can be seen from the examples given at the end.

1.3.1 Finite State Grammar-Parser

The first module in the UCSG architecture is a Finite State Grammar-Parser. The finite state grammar captures linear precedence, repetition and optional occurrence of words in word groups. It has been shown that these aspects are sufficient to recognize word groups and finite state machines are both necessary and sufficient for the purpose [93]. It is also well known that finite state machines are computationally efficient - linear time algorithms exist for recognizing word groups. Finite state grammars are also conceptually simple and easy to develop and test. It may be noted that detailed analysis of the internal structure of word groups (modifier-modified relationships, for example) is beyond the scope of the current system. There is thus no need for any hierarchical structure analysis at this stage.

In the UCSG Shallow Parsing Architecture, the Finite State Grammar is over-general by design. The goal is to accept *all* valid word groups but not necessarily the *only* word groups that are appropriate in context for a given sentence. Many additional word groups may be produced due to lexical ambiguities. We do not aim to restrict or prune the various possibilities. Instead we use a separate module to rate and rank these word groups.

The Finite State module accepts a sentence (either already POS tagged or tagged with all possible categories using a dictionary) and produces an unordered set of possible chunks taking into account lexical ambiguities if any.

1.3.2 HMMs for Rating and Ranking Chunks

The second module is a set of Hidden Markov Models (HMMs) used for rating and ranking the word groups produced by the Finite State Grammar. The hope is to get the best chunks near the top. This way, although we are not restricting to *only* the appropriate chunks in context, we can hope to get the right chunks near the top and push down others.

Words are observation symbols and POS tags are states in our HMMs. Formally, a HMM model $\lambda = (\pi, A, B)$ for a given chunk type can be described as follows:

Number of States (N) = number of relevant Categories

Number of Observation Symbols (M) = number of Words of relevant categories in the language

The initial state probability

$$\pi_i = P\{q_1 = i\} \tag{1.2}$$

where $1 \leq i \leq N$, q_1 is a category (state) starting a particular word group type.

State transition probability

$$a_{ij} = P\{q_{t+1} = j | q_t = i\} \tag{1.3}$$

where $1 \leq i, j \leq N$ and q_t denotes the category at time t and q_{t+1} denotes the category at time t+1.

Observation or emission probability

$$b_j(k) = P\{o_t = v_k | q_t = j\} \tag{1.4}$$

where $1 \leq j \leq N$, $1 \leq k \leq M$ and v_k denotes the k^{th} word, and q_t the current state.

While building HMMs, a manually checked and certified chunked corpus can be used if available. In this case, HMM parameters can be estimated right away. However, such labelled training data is rarely available. When no parsed corpus is available, we can rely on a POS-tagged corpus. In the latter case, a bootstrapping strategy is proposed to refine the HMM parameters. See figure 4.2. We first pass a large POS tagged corpus through the Finite State module and obtain all possible chunks. Taking these chunks to be equiprobable, we estimate the HMM parameters by taking the ratios of frequency counts. One HMM is developed for each major category of chunks, say, one for noun-groups, one for verb-groups, and so on. The B matrix values are estimated from a dictionary that includes frequency counts for each word in every possible category.

We simply estimate the probability of each chunk using the following equation :

$$P(O, Q|\lambda) = \pi_{q_1} b_{q_1}(o_1) a_{q_1,q_2} b_{q_2}(o_2) a_{q_2,q_3} \cdots a_{q_{t-1},q_t} b_{q_t}(o_t) \tag{1.5}$$

where q_1 ,q_2, $\cdots$, q_t is a state sequence, o_1 , o_2,$\cdots$, o_t is an observation sequence. Note that no Viterbi search involved here and the state sequence is also known. Thus even Forward/Backward algorithm is not required and rating the chunks is therefore computationally very efficient.

The aim here is to assign the highest rank for the correct chunk and to push down other chunks. Since a final parse is a sequence of chunks that covers the given sentence with no overlaps or gaps, we evaluate the alternatives at each position in the sentence in a left-to-right manner.

Here, we use *Mean Rank Score* to evaluate the performance of HMM modules. *Mean Rank Score is the mean of the distribution of ranks of correct chunks produced for a given corpus.* Ideally, all correct chunks would be at the top and hence the score would be 1. The aim is to get a Mean Rank Score as close to 1 as possible. We have used a manually parsed corpus of 4000 sentences to evaluate the Mean Rank Scores.

1.3.3 Parse Generation and Ranking

The third module is for identifying the best chunk sequence or global parse for a given sentence. It generates all possible parses hopefully in best first order. We can of course limit the number of parses generated if required but the ability to produce all possible parses is fundamental to the architecture. Note that we do not produces all possible parses first and then rate and rank them - the parse generation process inherently incorporates best-first search.

Choosing the locally best chunks at each position in a given sentence does not necessarily give us the best parse (chunk sequence) in all cases. HMMs are local to chunks and global information such as the probability of a chunk of a given type starting a sentence or the probability of a chunk of a particular type occurring next to a chunk of a given type are also useful. These probabilities can be obtained from a fairly small chunked corpus. We have used a best first search algorithm to get the best parse (chunk sequence) for a given sentence. The following steps describe how we map a given sentence and word groups present in the sentence into a graph and apply best first search.

- The positions in the sentence are treated as nodes of the resulting graph. If a sentence contains N words then the graph contains $N + 1$ nodes corresponding to the $N + 1$ positions in the sentence.

- Word group $W_{i,j}$ is represented as an edge form node i to node j.

- The probability of a word group $W_{i,j}$ given by HMM module and the transition probability from previous word group type to current word group type are combined to estimate the cost of an arc between the nodes i and j.

- We always start from the initial node 0. Length of the sentence N is the goal node.

Now our parse selection problem of a sentence containing N words becomes the task of finding an optimal path from node 0 to node N.

In best first search, we can inspect all the currently-available nodes, and rank them on the basis of our partial knowledge. Here high rank means that the node looks most promising in relation to the goal. At each step, we select the most promising of the nodes we have generated so far. We then expand the chosen node to generate it successors. If one of them is a solution, we can quit. If not, all those new nodes are added to the set of nodes generated so far. Again the most promising node is selected and the process continues. In the worst case, the best first search algorithm runs in exponential time because it expands many nodes at each level. In big-O notation, this is stated as $O(b^m)$, where b is the branching factor (i.e., the average number of nodes added to the open list at each level), and m is the maximum length of any path in the search space. Memory consumption is also a big problem apart from time complexity. The number of nodes that are stored in memory rapidly increases as the search moves deeper into the graph and expanding too many nodes can cause the algorithm to run out of memory.

Beam search is a heuristic search algorithm that is an optimization of best-first search. Like best-first search, it uses a heuristic function to estimate the promise of each node it examines. Beam search, however, only unfolds the first m most promising nodes at each depth, where m is a fixed number, the "beam width." While beam search is space-bounded as a function of m, it is neither optimal nor complete when m is finite. As m increases, beam search approaches best-first search in complexity.

It may be observed that linguistic knowledge has so far been restricted to finite state grammar and linguistic constraints should be expected to play an important role in parse generation and ranking. It is of course possible to incorporate a wide variety of other statistical and machine learning techniques for optimum chunk sequence selection. We would need a reasonable sized high quality chunked corpus for training. We have also explored A* best first search strategy.

1.3.4 Bootstrapping

The HMM parameters can be refined through bootstrapping. Since we need to work with large data sets running into many hundreds of thousands of sentences, Baum-Welch parameter re-estimation would not be very practical. Instead, we can use parsed outputs to re-build HMMs. It may be recalled that originally HMMs were built from chunks obtained from the over-general finite state parser taking all chunks as equi-probable. By parsing a given sentence using the system and taking the top few parses only as training data, we can re-build HMMs that will hopefully be better. We can also simply use the top-ranked chunks for re-building the HMMs. This would reduce the proportion of invalid chunks in the training data and hence hopefully result in better HMM parameters.

1.4 Summary of Results

In this work, we have built a dictionary that includes words, POS tags, frequency of occurrence for each tag for each word from the British National Corpus (BNC). We have developed a manually parsed corpus of 4000 sentences according to UCSG syntax by taking sentences from a wide variety of sources. We have shown that finite state machines are sufficient to produce all valid word groups for a given sentence. We have evaluated FSM module in terms of the number correct chunks it can recognize i.e recall. We have achieved a high recall of 99.5% on manually parsed corpus, 95.06% on CoNLL test data and 88.02% on Susanne corpus. We have successfully built HMMs by using only POS tagged BNC sentences and used them for rating and ranking word groups. We have evaluated the performance of the HMM modules in terms of mean rank score i.e. mean of the distribution of ranks of the correct chunks in the manually parsed corpus. We have obtained good mean rank scores i.e. 2.26 for plain sentences and 1.57 for POS tagged sentences on a test data of 4000 sentences.

We have proposed a best first search algorithm to select the best chunk sequence as a parse for a given sentence. All possible parses are produced but in a best first order. When we restrict best first module to give best five parses and time limit to 3 epoch seconds, we have obtained 45.52% correct parses within the top 5 for plain sentences and 68.02% of correct parses within the top 5 for POS tagged sentences. The percentage of correct chunks in the top parse is 78.70 for plain sentences and 78.42 for POS tagged sentences. We have also used a modified beam search method but here we may loose some of correct parses because of pruning although substantial speed up can be achieved. The number of correct parses in the top position for plain sentences has also increased from 28.25% to 31.55%. Number of sentences that can be parsed within the stipulated time increased from 54.67% to 100%.

We have proved our idea of bootstrapping to improve HMMs parameters as well as the performance of the whole parser. We have done bootstrapping in three ways: by taking HMM top ranked chunks, chunks from top parse given by third module and both combined. We have found that bootstrapping from top of parse of best first search module gives best results. We are able to improve the mean rank score to 2.21 from 2.26 in the first iteration and to 2.16 in the second iteration. The performance of the parser also improved in terms of pushing correct parses to the top. Before bootstrapping, there were 28.25% correct parses in the top position for plain sentences and this improved to 30.25%. The percentage of correct chunks in the top parse improved from 78.70% to 83.92%. For tagged sentences, the percentage of correct chunks in top parse improved from 78.42% to 88.26%. The percentage of correct parses in top position improved from 44.35% to 54.82%.

All the experiments have been carried out on a desktop PC with Pentium Core 2 DUO 1.86 GHz Processor and 1 GB RAM. The entire system has been implemented in Perl under Linux.

1.5 Main Claims and Contributions

1. *It is possible to develop wide coverage partial parsing systems for Natural Languages in a reasonable amount of time without need for a parsed training corpus to start with. Only a large POS tagged corpus is necessary.* [75].

This has been demonstrated for the case of English in this work.

2. *This can be achieved by a judicious combination of linguistic and statistical techniques.* In this thesis we have shown that a finite state grammar at chunk level combined with HMMs form a good combination. An architecture for wide coverage partial parsing has been proposed.

3. *Finite State Grammars with very high Recall can be built with relative ease at chunk level.* Finite State Grammars are easy to understand and visualize. Recognition with Finite State Grammars is computationally efficient - linear time algorithms exist. In this work, we have seen that very high Recall is achievable for English.

4. *The chunks produced by the system are somewhat more semantically oriented and closer to what is expected in a full syntactic parsing.* See examples at the end.

5. *Chunk level HMMs can be developed from a large POS Tagged corpus using the Finite State Grammar-Parser, without need for a parsed training corpus to start with.* Here we have developed HMMs from the British National Corpus of English and demonstrated their effectiveness.

6. *HMMs are used only for rating and ranking the chunks already obtained from the Finite State Grammar-Parser, not for recognizing chunks per se.* Since the chunks are already available and POS tags are also known for each word, even the Forward/Backward algorithm is not required and *evaluation can be done in linear time.*

7. *HMMs can produce good ranking, tending to push the correct chunks to positions near the top.* Good Mean Rank Scores have been achieved.

8. *HMM parameters can be refined by bootstrapping.* This has been demonstrated successfully in our bootstrapping experiments.

9. *A variety of Best First Search strategies can be employed to obtain globally best chunk sequences or parses for a given sentence..* In this work, we have shown how best first search strategy can be used to produce globally best parse. But the time and space complexity for best first strategy is exponential in nature as branching factor and sentence length increases. Hence, we have also proposed a modified beam search strategy to improve

the efficiency. Here we may lose some of the correct parses because of pruning and the results depend on size of the beam. Some work has also been done in applying the A* best first search algorithm. There is scope for incorporating more linguistic constraints here.

10. *It is possible to develop high quality (manually checked) parsed corpora using the system.* A 4000 sentence manually checked parsed corpus has been developed and used for development as also for testing and evaluation.

11. Large scale POS tagged corpora are available, or can be easily developed, for other languages of the world, including Indian languages. Indian languages are characterized by free word order and rich morphology. Nevertheless, words within chunks are order specific and thus Finite State Grammars at chunk level will not be much different. *The overall architecture should therefore be of interest in developing wide coverage partial parsing systems for Indian languages as well.*

12. *A wide coverage dictionary for English including frequency of occurrence of each word in each tag has been developed and found to be useful for chunk generation as well as rating and ranking using HMMs.*

13. *A Decision Tree solution for the sentence boundary detection problem has been developed and shown to give good performance.*

14. *A Language Identification system from small text samples for pair-wise language identification among 9 major Indian languages has been developed* using multiple linear regression as a two-class classification model. Good performance has been obtained.

15. *A variety of statistical analyses have been carried out on a nearly 40 Million word text corpus of Telugu. The results should be useful for further work on Telugu.*

1.6 Organization of the Thesis

The subsequent chapters of the thesis are organized as follows.

Chapter 2 presents summary of different grammar formalisms, different approaches for parsing and briefly describes various full and shallow parsing systems.

Chapter 3 briefly describes the Universal Clause Structure Grammar, a framework for parsing natural language sentences being developed at University of Hyderabad.

Chapter 4 describes the main contributions of this thesis. It describes different modules of the UCSG shallow parsing architecture in detail and also describes our methodology for building wide coverage shallow parsing systems by using judicious combination of linguistic and statistical approaches.

Chapter 5 presents a detailed analysis of our experiments and results carried out on different modules of the UCSG shallow parser for English. It also gives the comparison of UCSG shallow parser outputs with other shallow parsing systems available.

Chapter 6 summarizes the results of various modules in UCSG shallow parser and also describes our major claims and achievements in this work.

Chapter 7 summarizes directions for future research.

Appendix A presents the tag set used in UCSG finite state grammar.

Appendix B presents sample finite state grammar used in UCSG finite state machine.

Appendix C presents some of the examples from manually parsed corpus developed by us at University of Hyderabad. These examples are useful in understanding the ideology in developing manually parsed corpus and also the quality of the manually parsed corpus.

Appendix D presents the list of publications during this Ph.D work.

Chapter 2

Background and Literature Survey

2.1 Introduction

Parsing means "*analyzing syntactic structure in a sentence according to a formal grammar*" [33]. The word *Syntax* has originated from the Greek words [64] *(syn, means, "co-" or "together") and (taxis, means "sequence, order, arrangement").* Syntax can be described in linguistic terms as the study of the rules, or "patterned relations" that govern the way the words in a sentence come together. It concerns how different words are combined into phrases, phrases into clauses, which, in turn, are combined into sentences. A syntactic analysis of a sentence helps us to determine the meaning of a sentence from the meaning of its words.

Parsing is a step towards the final goal of natural language understanding or generation. Parsing is a prerequisite to many tasks involving human language computing, both because a significant part of the meaning of a sentence is encoded in its grammatical structure, and because models that ignore structure are insufficient to distinguish well-formed from ungrammatical utterances. Applications that potentially benefit from syntactic parsing include question answering, rule-based automatic translation, information extraction and summarization. For example, in Information Extraction and Question Answering, researchers will be interested in information about some specific syntactic or semantic relations such as agent, object, location, time (- basically, who did what to whom, when, where and why).

Grammar and lexicon play vital roles in performing syntactic analysis. The grammar and lexicon of a language together define which structures are possible in that language and which words may go where in that structure.

2.1.1 Grammars

Languages manifest as linear sequence of symbols in text or speech form. Not all sequences of these symbols are meaningful. We need to restrict the set of all possible sequences of symbols and include *all* the valid sequences and *only* the valid ones. In other words, we need to impose constraints on the possible sequences of symbols. Such a set of constraints, expressed as rules, or principles or whatever, is called a grammar. A grammar of a language is a formal specification of the valid structures in that language. Thus we may think of grammars at various levels - morphology is the grammar at word level, syntax is usually concerned with grammars at the sentence level, discourse grammars define valid discourse elements and so on.

In the 1950s, there were major developments taking place in the field of computer science, including the development of high level programming languages. Around the same time Noam Chomsky laid out the foundations of formal properties of grammars. Chomsky was one of the pioneers in identifying the correspondence between the different types of grammars and the formal computational models that are required to recognize them. Chomsky classified grammars based on string rewriting rules into four classes of formal complexity:

- Type-0 Grammars are unrestricted grammars, correspond to recursively enumerable languages, require Turing Machines to recognize them
- Type-1 Grammars are context-sensitive grammars, correspond to context-sensitive languages and require a type of automata called linear-bounded automata to recognize them
- Type-2 Grammars are context-free grammars, correspond to CFLs and require PDAs to recognize them.
- Type-3 Grammars are regular grammars, correspond to regular languages and require FSAs to recognize them.

From the Chomsky hierarchy, we can observe that regular grammars are very simple and finite state machines can be used to generate these languages. Recognition of strings using a DFA can be done with linear time complexity. Of course there are limitations to what they can do. For example, it is not possible to write an FSA that generates the language $a^n b^n$, i.e. the set of all strings which consist of a (possibly empty) block of a-s followed by a (possibly empty) block of b-s of exactly the same length. That is FSAs have certain expressive weaknesses. This also limits their expressive adequacy from a linguistic point of view, because many linguistic phenomena can only be described by languages which cannot be generated by FSAs. Regular grammars can handle all finite languages and also some infinite languages characterized by repeated substrings. Natural languages are commonly taken to be infinite, they involve arbitrarily deeply nested structures, not mere repetition of subparts. Regular grammars are not sufficient to capture the whole of natural languages[96].

However, there are linguistic applications where the expressive power of finite state methods seems to be sufficient. The flip side of the coin is that they usually behave very well computationally and the flop side is their expressive power is weak. If one can find a solution based on finite state methods, the implementation will probably be efficient. Areas where finite state methods have been shown to be particularly useful are phonological and morphological processing. But finite state methods have also been applied to syntactic analysis. Although they are not expressive enough for a full syntactic analysis, there are many applications where a partial syntactic analysis of the input is sufficient. Such partial analyses can be constructed with cascades of finite state automata where one machine is applied to the output of another. Furthermore, Hidden Markov Models, which are very common for speech recognition and part of speech tagging, can be seen as a variant of FSAs which assigns probabilities to its transitions.

Context free grammars are powerful enough to describe the syntax of most programming languages; in fact, the syntax of most programming languages are specified using context-free grammars. On the other hand, context free grammars are simple enough to allow the construction of efficient parsing algorithms which, for a given string, determine whether and how it can be generated from the grammar. Earley parser is an example of such an algorithm, while LR and LL parsers only deal with more restrictive subsets of context-free grammars.

Context Free Grammars:

A Context Free Grammar is a 4-tuple (N,T,P,S) where N is a finite set of non-terminal symbols, T is a finite set of terminal symbols, disjoint from N, S is a special designated symbol from N called the Start Symbol, and P is a finite set of Production Rules (or Productions), of the form $A \rightarrow \alpha$

where A is any non-terminal symbol and α is any sequence of terminal and non-terminal symbols. Unless the language itself contains the empty string, α can be required to be non-empty. These grammars are called 'context free' because all rules contain only one symbol on the left hand side and wherever we see that symbol while doing a derivation, we are free to replace it with the stuff on the right hand side. That is, the 'context' in which a symbol on the left hand side of a rule occurs is unimportant and we can always use the rule to make the rewrite while doing a derivation.

The language generated by a context free grammar is the set of terminal symbols that can be derived starting from the start symbol 'S'. A language is called context free if it is generated by some context free grammar. Not all languages are context free in nature. For example, $a^n b^n c^n$ is not. No CFG can do the job.

There is a more fundamental aspect of CFGs, namely as tree admissibility rules. A parse tree is a finite tree all of whose interior nodes (that is, nodes with daughters) are licensed or admitted by a grammar rule. First, it is linguistically most fundamental representation of natural language sentences. Thus CFG rules tell us which tree structures we have for a given sentence. Second, the idea of parse trees brings us to an important concept: ambiguity. A string is ambiguous if it has two distinct parse trees.

The next question would therefore naturally be whether context free grammars can be used to parse natural languages. There are deterministic subclasses of CFGs which can be parsed in deterministic linear time. Even parsers based on general CFGs have a worst case time complexity of $O(n^3)$ where 'n' is the number of words in a sentence. Thus parsing with CFGs can be quite efficient. In the sixties, CFGs were actually widely applied to the study of natural languages.

Context-free grammars were very popular as models for natural language in spite of formal inadequacies of the model for handling some of the features that occur in natural languages.

The question of context freeness of natural languages has received a lot of attention. There have been many papers arguing for or against context freeness of natural languages. We need to clearly understand which aspects of the syntax of natural languages are really context free and which aspects are not. We will then be able to apply more powerful grammars only where they are really required. Parsing with CFGs will be very much faster than parsing with more complex grammars. CFGs have to be used for whatever aspects they are necessary and sufficient.

In a formal sense CFGs have sufficient weak generative capacity to deal with almost all aspects of natural languages. Natural languages are largely context free. In a practical sense, however, CFGs have several disadvantages. The following are the limitations of CFGs, when we want to use them for natural language parsing[96]:

1. **Functional Dependencies:**

   Languages enforce certain dependencies between the constituents in a sentence. Examples of this are grammatical agreement requirements and selectional restrictions. Thus 1) is grammatical but 2) is not, 3) is grammatical while 4) is not.

   1) The students write the test

   2) *The student write the test

   3) The student read the paper

   4) *The paper read the student

   If we have to enforce these dependencies using (only) CFGs, we will be faced with two problems. For one thing, we will be forced to introduce new categories since CFGs, like all other phrase structure rules, can only generate strings of terminal symbols. There would be singular-nouns and plural-nouns, singular-verbs and plural-verbs. There will be nouns that denote *things that can read* and nouns that denote *things which can not be read.* This would cause an explosion of rules leading to very large rule sets.

There would be separate rules for singular sentences and plural sentences. Computationally the syntactic system becomes extremely inefficient. Secondly, and more importantly, a simple requirement like agreement between subject and verb, which can be given in one simple statement in English, is being turned into a multitude of otherwise unrelated categories and rules. Simple generalizations are lost.

A word should be associated with a particular category purely based on its own intrinsic properties. A grammatical relation between two independent words is an intrinsic property of neither of these words. The phrase structure rules have no direct way of specifying these relations. Thus CFGs, why, all types of phrase structure rules for that matter, are unsuitable for dealing with dependencies between different constituents in a sentence. Also, the structural descriptions which CFG rules naturally generate, namely trees, cannot depict functional dependencies directly. It should be emphasized that we are not asserting that CFGs are insufficient. CFGs do have the necessary generative capacity to generate valid and only valid strings as far as such dependencies are concerned. But that is simply not the right way of doing things. Functional dependencies must be separately and explicitly specified in the grammar and clearly depicted in the structural descriptions. CFGs and trees are not the best way to do this.

2. **Relatively Free Word Order Languages:**

   Linear order of words and phrases in a sentence may be significant - changing the order may render the sentence ungrammatical or anomalous. A sentence is a sequence of words in English and it looks almost unimaginable to view a sentence as an unordered set of words. However, there are languages of the world where order of words is the least important aspect of structure. All permutations of words in a Sanskrit sentence are grammatically valid [96] and mean exactly the same thing. A sentence could be viewed as a set of words, not really a sequence.

   There are also a number of human languages where there is considerable, though not unlimited scope for changing the order of words in a sentence without significantly altering its basic meaning (to be more precise, without altering the functional structure of the sentence). Modern Indian languages are examples of this. All of the following Telugu sentences mean essentially the same thing [96].

1)

raamuDu	baDiki	siitatoo	veLLaaDu
Rama	school-to	Sita-with	went

2)

baDiki	raamuDu	siitatoo	veLLaaDu
school-to	Rama	Sita-with	went

3)

siitatoo	raamuDu	baDiki	veLLaaDu
Sita-with	Rama	school-to	went

The strong point of CFG is that it can effectively deal with both the linear and hierarchical structure inherent in human languages. This ability to deal with linear structure comes back as a weakness when we have to parse sentences in relatively free word order languages. Here a variety of word orderings are allowed and *linear order is not to be considered significant*. CFGs however, have no way of getting rid of their hold on linear order. A CFG rule cannot assert the hierarchical structure alone, without implying the order of constituents.

3. **Long Distance Dependencies and Movement:**

   Linear order of words and phrases in a sentence may be significant - changing the order may render the sentence ungrammatical or anomalous. There are valid syntactic constructions, however, where there are deviations from the usual order. Therefore these formalisms based on CFGs need to posit a *normal* or unmarked order and view syntactic constructs which deviate from the normal order as involving *movement*. The grammar is expected to specify what constituents move from which place to which place and under what conditions.

Some movements are local or bounded, as in the case of auxiliary shift in simple yes-no questions (example 11) and subject-object inversion in passives (example 12).

11) *Can* the company sell Pentiums now?

12) *My Pentium* was replaced by *someone*.

Other types of movements take place across arbitrarily long distances and are termed *long distance dependencies*. Relative clauses (examples 13 and 14) and wh-questions (examples 15, 16, and 17) in English are examples. In these constructions a constituent from within a clause will have been removed from its normal position and moved to a different position. Long distance dependencies have been one of the serious problems for syntactic systems and various grammar formalisms have developed special techniques for dealing with them.

13) The machine which the company sold __ to me was defective

14) The machine which the company, which I believe everyone trusts, sold __ to me was defective

15) Which computers does the company sell __?

16) Which computers does the company which I believe everyone trusts sell __?

17) Which computers do you think the company which I believe everyone trusts sells __?

Historically, the concepts of movement and other kinds of transformations were introduced in order to account for the fact that sentences superficially looking different often have roughly the same meaning. So a *deep* structure, something closer to meaning than the surface structure of a sentence, was taken as the starting point and other related structures were obtained through transformations. It was soon shown that many of the supposedly meaning preserving transformations did in fact risk meaning changes and linguists had to give up the idea of linking deep structure to meaning. Linguists forgot the meaning aspect but stuck to the notions of deep and surface structures, now renamed D-Structure and S-Structure, and movement continued to be the primary mechanism of going from the D structure to the S structure. In the more recent minimalistic approach, the concepts of deep and surface structures have been completely abandoned with.

The notions of movement and other transformations are direct consequences of the employment of phrase structure rules and the corresponding ordered trees for describing the syntax of natural languages. As we have already seen, a tree is a mess of different levels of meaning units - words, phrases, clauses and sentences. In terms of linear position, the parts of a single meaning unit can be widely spaced apart. Long distance dependencies are only one type of functional dependencies and CFGs cannot effectively deal with any kind of dependency between two constituents.

If we could view sentences as sets, rather than as sequences of words (or phrases or clauses), there would be no question of any distance, long or short. Meaning of sentences is not best described in terms of length or distance, whether expressed in terms of number of intervening structures, words or in inches or meters. It looks like we are trying to solve non-existing problems.

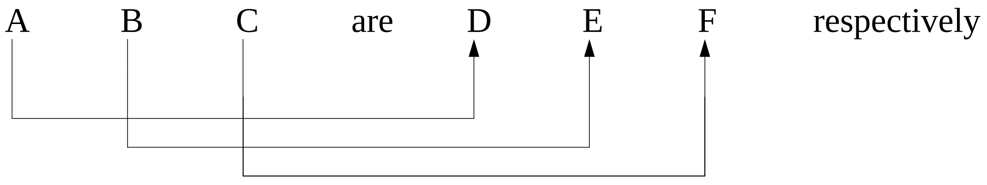
Western grammar formalisms have been strongly influenced by languages like English where linear order constraints are very significant. Linear order has become such an all important and dominating core aspect of western linguistic models that these models have to bring in highly unnatural and unnecessary complications to deal with languages where word order is least important. If you want to work with Indian languages, the first thing you must learn is to unlearn this obsession with phrase structure rules and tree structures. Forget about word order. Forget about movement. Forget about distance. Forget trees.

4. **Cross Serial Dependencies:**

CFGs can handle constituents that come linearly one after the other as well as constituents that are properly nested one inside the other. By proper nesting we mean that an inner constituent must end before the outer constituent closes. That is a last-in, first-out property must be satisfied. In English, we find some examples of cross serial dependencies as illustrated below.

18) Manmohan Singh, A P J Abdul Kalam, and Rajasekhara Reddy are the Prime minister of India, the President of India and the Chief Minister of Andhra Pradesh respectively.

The 'respectively' construct 'A, B, C are D, E, F respectively' links A to D, B to E and C to F in a cross serial manner:



Cross serial dependencies occur in programming languages too. One example is in the requirement that the arguments must match one-to-one in a function definition and function call. Compilers simply ignore this aspect while doing the syntactic analysis of the source program and postpone the checking to a later phase. We could do the same thing for constructs like 'respectively' in English by simply taking 'A,B and C' and 'D, E and F' to be single composite constituents. It would be alright to postpone a few complex aspects to a later stage of analysis. The only thing a CFG grammar will not be able to do is to identify the links between the items in the two sets, rest of syntactic analysis can go on without any problem. The 'respectively' construct is perhaps the only example of cross-serial nesting in English. This is not a major problem.

There are languages such as Dutch, however, where we come across an infinite set of grammatically correct sentences with cross serial nesting. A way of directly dealing with them would therefore be preferable.

If universality in the theoretical sense is not the main concern and the set of languages we wish to deal with do not show cross-serial dependencies in any serious measure, we may still go ahead and claim that CFGs are good enough in this limited sense of generative capacity.

5. **Unbounded Branching**

CFGs capture repetition through recursion. In many situations what we want is mere repetition and not recursion. For example, items in conjunction should really be all at the same level. Since there is no *a priori* bound on the number of items in conjunction, either we have to use recursive rules or make the number of rules potentially infinite. A sequence of a's can only be captured by one of the recursive rules

A → a A

or

A → A a

along with the base rule

A → a

or by the potentially infinite set of rules

A → a

A → a a

A → a a a

and so on. Computational grammar formalisms require that the rule sets be finite and hence in general imposing nonexistent hierarchical structure through recursive rules is inevitable. CFG rules are suitable for dealing with recursion but not for handling repetition. It is worth noting that regular expressions, equivalent in computational complexity to the much simpler regular grammars, can directly and effectively deal with repetition.

2.1.2 Grammar Formalisms

Every grammar defines a language but a given language can be generated by many different grammars. All possible grammars of a language will not in general be equally simple, natural and amenable for efficient parsing. Also, grammars can be expressed in a variety of ways. Grammars can be formulated as regular expressions, finite state automata, phrase structure rules, trees and tree operations, principles, constraints, etc.

A Grammar formalism is a meta language that specifies the kinds of grammars that can be written, defines the structure of grammars and hence the formal properties of the syntactic system employing those grammars. A Grammar formalism defines languages by specifying a set of constraints that characterize the set of well-formed structures. Parsing systems generally use the suitable grammar formalism to perform syntactic analysis of the natural language text.

A variety of grammar formalisms have been proposed for parsing: Case Grammars [43], Government Binding theory [25, 130], Lexical Functional Grammar [66, 65, 130], Definite Clause Grammars [1], Functional Unification Grammar(FUG) [67], Generalized Phrase Structure Grammar [44], Tree Adjoining Grammar [51, 62, 132], Dependency Grammars [57], Categorical Unification Grammar(CUG) [38] , Head Driven Phase Structure Grammar(HPSG) [111], Link Grammar [50, 134], PATR, Probabilistic Context Free Grammar [24, 68], Minimalist Approach [148, 100], Paninian Grammar [11] etc.

One of the wide-spread class of linguistic formalisms are the so-called constraint-based grammar formalisms which are also often subsumed under the term unification grammars. The key characteristic of constraint based grammar formalisms is the use of feature terms(sets of attribute-value pairs) for the description of linguistic units, rather than atomic categories as in phrase-structure grammars. Unification based grammars make use of feature structures (FS) to represent lexical properties, syntactic constraints. Feature Structure is a Directed Graph, in which all the nodes and edges have an associated name. The name associated with a node is called Type and the name associated with a edge is called Feature. The types of edges or attributes that can be associated with a node are determined uniquely by the Type. This uniform representation (using FSs) drastically reduces the number of operations in a parser. These feature structures are manipulated by the operation of unification. Because feature-terms can contain a lot more information than atomic categories, unification-based theories went hand-in-hand with lexicalization trend i.e. information that earlier was part of the grammar proper got integrated in the lexicon. This family includes LFG, Categorial Grammar, GPSG, HPSG and TAG.

2.1.2.1 Lexical Functional Grammar

Lexical functional grammar (LFG) was initiated by Joan Bresnan and Ronald Kaplan [66, 58] in the 1980s. LFG uses the following two different structures for representing different levels of linguistic information about a sentence:

- Constituent Structure (C-structure) – C-structure captures language-specific phenomena such as word order and the grouping of constituents into larger phrases in the form of context-free trees. The C-structure in LFG represents the external structure of a sentence in the form of a phrase structure tree. It shows the syntactic categories and both the linear order of constituents and the hierarchical grouping of words in a sentence. The hierarchical grouping of words in a sentence is governed by phrase structure rules which are in the form of the context-free grammar rules. As C-structure encodes surface syntactic information like word order and phrasal structure, it is language dependent. Constituent structures roughly corresponds to the PF in GB theory and to surface structure in GPSG.

- Functional Structure (F-structure) – This includes the representation of the higher syntactic and functional information of a sentence. The higher syntactic and functional information of a sentence is represented in F-structure as a set of attribute-value pairs. These pairs form the nodes of the acyclic graph structure. In an attribute-value pair of an F-structure, the attribute corresponds to the name of a grammatical symbol (e.g. NUMB, TENSE) or a syntactic function (e.g. SUBJ, OBJ,PRED,COMP,ADJUNCT etc.) and the value is the corresponding feature possessed by the concerning constituent. The value for each attribute can be an atomic symbol, a semantic form or a subsidiary F-structure. The functional information of a sentence includes the information about functional relations between parts of sentences and how each part of the sentence affects each other. The relationship between some elements of a sentence is shown in an F-structure by means of the links drawn between them. F-structure also expresses the information about the kind(s) of syntactic functions that each predicator (e.g. verb or preposition) governs.

According to LFG theory, C-structures and F-structures must satisfy certain formal well-formed-ness conditions. A C-structure/F-structure pair is a valid LFG representation only if it satisfies the Non-branching Dominance, Uniqueness, Coherence and Completeness conditions[66]. Non-branching Dominance demands that no C-structure category appears twice in a non-branching dominance chain; Uniqueness asserts that there can be at most one value for any attribute in the F-structure; Coherence prohibits the appearance of grammatical functions that are not governed by the lexical predicate; and Completeness requires that all the functions that a predicate governs appear as attributes in the local F-structure. The first three conditions (Non-branching Dominance, Uniqueness and Coherence) are monotonic, in the sense that if they are unsatisfied by a substructure they will also be unsatisfied by any superstructure. The Completeness condition, on the other hand, is non-monotonic in that larger structures may satisfy this condition while their substructures may not[66].

LFG uses the single operation of unification instead of the two separate operations of setting and checking the values of the feature dimensions as in ATN. This makes the LFG grammar somewhat easier to write. However, unification in LFG implies that a large number of feature structures are built and then rejected. Also unification itself is a costly operation. LFG also takes linear position too seriously and thus fails to deal effectively with relatively free word order languages.

2.1.2.2 Head-Driven Phrase structure Grammars

The Head-driven phrase structure grammar (HPSG) is an immediate successor to Generalized Phrase Structure Grammar (GPSG) which was developed by Carl Pollard and Ivan Sag[111] in 1987. The notion of the head constituent of a phrase is of central importance in HPSG.

HPSG relies on two essential components: (i) the formalism is based on lexicalism. This means that the lexicon is more than just a list of entries, it is in itself an explicit, highly structured representation of grammatical categories, encoded as typed feature structures; (ii) a set of descriptive constraints on the modeled categories expressing linguistic generalizations and declaratively characterizing the expressions admitted as part of the natural language. From a linguistic perspective, the set of descriptive constraints expressing the theory of an HPSG grammar consist of a) a lexicon licensing basic words, b) lexical rules licensing derived words, c) immediate dominance schemata licensing constituent structure, d) linear precedence statements constraining constituent order, and e) a set of grammatical principles expressing generalizations about linguistic objects.

The basic type HPSG deals with is the sign. Words and phrases are two different subtypes of sign. A word has two features: PHON (the sound, the phonetic form) and SYNSEM (the syntactic and semantic information), both of which are split into sub-features. Signs and rules are formalized as typed feature structures.

The Attribute-Value Matrix (AVM) describes objects of type word. The phon attribute here is simply taken to be a list of strings serving as a placeholder for an actual phonological representation for HPSG, as developed by Bird and Klein (1994) and Hohle (1999). The morpho-syntactic information which characterizes local properties of linguistic expressions are specified under the category feature which, along with the semantic content, are identified by the local part of the synsem value. The subset of category properties which are necessarily shared between mother and head daughter in a local tree are packaged together under the head feature. The valence feature specifies the combinatory potential of lexical items as lists of synsem objects (as opposed to lists of signs). Thus neither phonological information (specified in phon), nor the daughters feature, which we will see as encoding constituent structure in objects of type phrase, can be selected for, incorporating the well-supported generalization that syntactic selection is independent of phonological form and is consistently local.

2.1.2.3 Government-Binding theory

Government and binding[25] is a theory of syntax in the tradition of transformational grammar developed by Noam Chomsky. The name refers to two central sub-theories of the theory: government, which is an abstract syntactic relation, and binding, which deals with the referents of pronouns, anaphores, and R-expression. GB was the first theory to be based on the principles and parameters model of language, which also underlies the later developments of the Minimalist

Program.

GB theory is a set of sub-theories consisting of principles and parameters[31]:

- X-bar syntax describes the structure of phrases
- Projection Principle: it requires all the levels of syntax to observe the specifications for each lexical item given in its entry in the lexicon.
- Movement: it is relationship between two levels
    1. d-structure where the underlying structure is given
    2. s-structure where the related form of the sentence after movement is described, including traces of the underlying positions of the items
- Bounding theory prevents the relationship of movement from extending too far in the sentence
- Θ-theory deals with the assignment of semantic roles, constrained by Θ-criterion
- Case Theory assigns cases to Noun Phrases in the sentence
- Binding Theory concerns with the reference relationships of Noun Phrases
- Control Theory deals with the subject of infinitival clauses
- The Phonetic Form(PF) Component interprets s-structure to represent it as sounds
- The Logical Form(LF) Component represents the sentence as syntactic meaning

2.1.2.4 Tree Adjoining Grammar

Tree-adjoining grammar (TAG)[63, 61] is a grammar formalism defined by Aravind Joshi. Tree-adjoining grammars are somewhat similar to context-free grammars, but the elementary unit of rewriting is the tree rather than the symbol. It takes trees assigned to individual words as the core element of language representation – rather than linear rules or pairs of structures – along with a restricted definition of tree combination. Structural representations are built up from pieces of so-called elementary trees, which are taken as atomic. These trees can be combined by using one of two operations: Substitution and Adjoining. TAG thus constitute a tree-generating system, rather than a string-generating system as context-free grammar.

Tree Adjoining Grammar (TAG): A tree adjoining grammar G is a quintuplet ($\sum$,NT,S,I,A) where $\sum$ and NT are a finite set of terminal symbols and a finite set of nonterminal symbols, respectively, S is a distinguished nonterminal symbol called the start symbol, and I and A are a finite set of initial trees and a finite set of auxiliary trees, respectively.

An initial tree is a tree of which the interior nodes are all labelled with non-terminal symbols, and the nodes on the frontier are either labelled with terminal symbols, or with non-terminal symbols, which are marked with the substitution marker($\downarrow$).

An auxiliary tree is defined as an initial tree, except that exactly one of its frontier nodes must be marked as foot node (‘*’). The foot node must be labelled with a non-terminal symbol which is the same as the label of the root node.

According to Schabes et. al. a grammar said to be ‘lexicalized’ if it consists of 1) a finite structures associated with each lexical item serving as ‘head’ 2) an operation or operations for composing the structures. These structures specify extended domains of locality over which lexical constraints may be stated. In LTAG, each elementary tree contains at least one frontier node labelled with a terminal symbol. Thus each elementary tree is associated with at least one lexical element.

Tree-adjoining grammars are often described as “mildly context-sensitive”, meaning that they possess certain properties that make them more powerful (in terms of weak generative capacity) than context-free grammars, but less powerful than context-sensitive grammars as defined in the Chomsky hierarchy. Mildly context-sensitive grammars are (it is conjectured) powerful enough to model the grammars of natural languages while remaining efficiently parsable in the general case. TAG has been shown to be sufficient to handle both sub-categorization dependencies and filler-gap dependencies. In fact TAG grammars can also deal with crossed dependencies which CFG cannot handle. TAGs permit polynomial time parsing with a worst case time complexity of $O(n^4)$, just n times worse than CFGs at their worst.

2.1.2.5 Case Grammars

Case Grammar is a theory of grammatical analysis, developed by the American linguist Charles J. Fillmore [43] in 1968. This theory proposes to analyze sentences as constituted by the combination of a verb plus a set of deep cases (i.e. semantic roles), such as Agent, Location or Instrument.

According to Fillmore, each verb selects a certain number of deep cases which form its case frame. Thus, a case frame describes important aspects of semantic valency of verbs, adjectives and nouns. Case frames are subject to certain constraints, such as that a deep case can occur only once per sentence. Some of the cases are obligatory and others are optional. Obligatory cases may not be deleted, at the risk of producing ungrammatical sentences. For example, *Mary gave the apples* is ungrammatical in this sense.

A fundamental hypothesis of case grammar is that grammatical functions, such as subject or object, are selected in dependence of deep cases. Fillmore puts forwards the following hierarchy for an universal subject selection rule:

$Agent < Instrumental < Objective$

That means that if the case frame of a verb contains an agent, this one is realized as the subject of an active sentence; otherwise, the deep case following the agent in the hierarchy (i.e. instrumental) is promoted to subject.

The influence of case grammar on contemporary linguistics has been significant, to the extent that numerous linguistic theories incorporate deep roles in one or other form, such as the so-called Thematic Structure in Government and Binding theory. It has also inspired the development of frame-based representations in AI research.

2.1.2.6 Dependency Grammars

Dependency grammar (DG) is a class of syntactic theories developed by Lucien Tesniere[138]. Dependency Grammars describe the structure of a sentence in terms of binary head-modifier (also called dependency) relations on the words of the sentence. A dependency relation is an asymmetric relation between a word called head (governor, parent), and a word called modifier (dependent, daughter). A word in the sentence can play the role of the head in several dependency relations, i.e. it can have several modifiers; but each word can play the role of the modifier exactly once. One special word does not play the role of the modifier in any relation, and it is named the root. The set of the dependency relations that can be defined on a sentence form a tree, called the dependency tree.

There are several different versions of this grammar including Meaning Text Theory by Melcuk in 1988, Word Grammar by Hudson in 1990, Operator Grammar by Zellig Sabbetai Harris, Functional Dependency Grammar by Jarvinen and Tapanainen in 1997 and Extensible Dependency Grammar (XDG) by Ralph Debusmann in 2003. Link grammar is similar to dependency grammar, but link grammar includes directionality in the relations between words, as well as lacking a head-dependent relationship.

Dependency grammars, like phrase structure grammars, use trees (directed acyclic graphs) in order to depict the structure of a given phrase or sentence. While a phrase structure grammar associates the nodes in the tree with larger or smaller constituents and uses the arcs to represent the relationship between a part and the whole, all nodes in a dependency tree represent elementary constituents and the arcs denote the direct syntagmatic relationships between such elements.

Dependency grammars are not defined by a specific word order, and are thus well suited to languages with free word order.

2.1.2.7 Categorial Grammar

Categorial grammars were first proposed by K. Ajdukiewicz[5], and modified by many people including Y. Bar-Hillel[8] and Steedman[135] etc.

A categorial grammar has two components: lexicon and combinatory rules. Lexicon associates each word with a syntactic and semantic category. All information about possible syntactic combinations of constituents is encoded in their categories. The combinatory rules allow functions and arguments to be combined. There are two types of categories: functors and arguments. Arguments are like Nouns, have simple categories like N. Verbs and determiners act as functors. Instead of phrase structure rules, the grammar contains one or, in some formalisms, two combination rules that combine a functor and an argument by applying the function encoded in the functor to the argument constituent. The simplest combination rules are like X/Y or X\Y. X/Y means a function from Y to X i.e. something which combines with a Y on its right to produce X and X\Y with a Y on its left to produce and X. Modern categorial grammars include more complex combinatory rules which are needed for coordination and other complex phenomena and also include composition of semantic categories as well as syntactic ones.

The CCG proposed by Steedman[135] has the following two principles as "central theoretical assumptions". The principle of adjacency states that combinatory rules may apply to finitely many phonologically realized string-adjacent entities, and serves to rule out theoretical constructs such as empty categories. The principle of categorial government states that both bounded and unbounded syntactic dependencies are entirely determined by lexical syntactic types, which specify semantic valency and canonical constituent order, and nothing else. This amounts to a requirement that all information regarding the potential for syntactic dependencies is projected from the lexicon. Lambda abstraction is used as a notation for representing semantic interpretations and semantic composition is constrained by the principle of combinatory transparency. This principle requires that the syntactic form of a combination rule completely determines its semantic form.

Categorial grammars of this form (having only function application rules) are equivalent in generative capacity to context-free grammar and are thus often considered inadequate for theories of natural language syntax. Unlike CFGs, categorial grammars are lexicalized, meaning that only a small number of (mostly language-independent) rules are employed, and all other syntactic phenomena derive from the lexical entries of specific words.

These grammars can also be applied to free word order languages.

2.2 Parsing Strategies

Various parsing strategies have been proposed to produce parse trees such as top-down, bottom-up, depth-first, breadth-first, and chart parsing.

Top-down and bottom-up are rival solutions that have been proposed as alternative solutions for the strategy question regarding the direction of the parsing process. Top-down parsing begins with the start symbol (usually a sentence S) and applies the grammar rules forward until the symbols at the terminals of the tree correspond to the components of the sentence being parsed. The parser could recursively continue in this fashion until it arrives entirely at terminal symbol states, and then it could check the input sentence to see if the classes of words in it matched with the rewritten sequence of terminal symbols. Parsers developed using this approach are called top-down parsers. It takes a rule, the left-hand side of which is a start symbol, rewrite this to its right hand symbols. Then it expands the left most nonterminal symbol out of these and so on until the input words are generated. Top-down parser are also known as recursive-descent parsers or LL parsers.

Bottom-up parsing starts from each word and assigns its grammatical category until it reaches the start symbol. This operation is repeated, at each state, using the sequence of highest-level labels as the new string to operate on. It continues until a single tree whose terminals are the words of the sentence and whose top node is the start symbol (usually S, for sentence) has been produced. The task of the parser would now appear to be that of attempting to group words into their respective categories together in a manner permitted by the grammar. The parser pushes words onto the stack until the right-hand side of a production appears as one or several topmost elements of the stack. These elements may then replaced by the left-hand side category of the production. If successful, the process will continue by successive shift and reduce step until the stack have been reduced to the start category and the input is exhausted. These bottom-up parsers are also known as table-driven or shift-reduce or LR parsers.

Top-down methods have the advantage of being highly predictive. This means that a word might be ambiguous when considered in isolation, but if some of those grammatical categories cannot be used in a legal sentence, then these categories may never even be considered. These parsers are easy to code and can tokenize quickly. However, this method has a duplication of effort that becomes a serious problem, and large constituents may be rebuilt again and again as they are used in different rules. This makes it slow in backtracking and recursion can be expensive. It can not handle left recursion.

In contrast, the bottom-up parser only checks the input sentence once, and only builds each constituent exactly once. The basic observation to make about the bottom-up parser is that it works from left to right: it does everything it can with the first item before exploring what it can do with the next two items, and so on. That is, the parser is bottom up, driven entirely by the data presented to it and building successive layers of syntactic abstraction on the basis of data provided. It handles left recursion. The code may be cumbersome and tail recursion will be handled poorly.

Unfortunately, whether one chooses top-down or bottom-up to implement, the payback is prohibitively expensive, as the parser would tend to try the same matches again and again, duplicating much of its work unnecessarily. Hence, to avoid such reduplication problem there should be a mechanism that allows parser to store results of the matching it has done so far. Such a technique is called chart-based parsing. There are three well known dynamic programming parsers: the Cocke-Younger-Kasami algorithm (CYK), the Graham-Harrison-Ruzzo (GHR) algorithm and the Earley parser.

A chart is simply a data structure for storing complete and incomplete constituents of parsing process in such a way that they can be reused later on. That is, it stores the intermediate results and maintains the record of rules that have matched but are not completed. Recording of intermediate results is a form of dynamic programming that avoids duplicate work. Without a means of storing such information, depth-first search involves undoing and rebuilding constituents, breadth-first search needs to store such information anyway, and a chart avoids duplication of constituents which are shared between parses. Chart parsing is also very flexible as regards parsing because search strategy can use it to parse bottom-up or top-down or can use it for depth-first or breadth-first search.

Breadth-First search algorithm of a graph explores all nodes adjacent to the current node before moving on whereas Depth-First search algorithm of a tree explores the first child of a node before visiting its siblings. The advantage of breadth-first search is that it prevents us from zeroing in on one choice that may turn out to be completely wrong; this often happens with depth-first search, which causes a lot of backtracking. Its disadvantage is that we need to keep track of all the choices and if the bag gets big, we have to pay a computational price.

2.3 Brief Survey of Parsers based on Linguistic Grammar Formalisms

Here we give a brief description of some of the parsers that are based on linguistic grammar formalisms.

2.3.1 Link Grammar Parser

The Link Grammar Parser[134] is a syntactic parser of English, based on a formal grammar called a link grammar. Given a sentence, the system assigns to it a syntactic structure, which consists of a set of labelled links connecting pairs of words. The parser also produces a "constituent" representation of a sentence (showing noun phrases, verb phrases, etc.).

Link grammar has roughly seven hundred definitions that captures many phenomena of English grammar. It handles: noun-verb agreement, questions, imperatives, complex and irregular verbs, different types of nouns (mass nouns, those that take to-phrases, etc.), past- or present-participles in noun phrases, commas, a variety of adjective types, prepositions, adverbs, relative clauses, possessives, and many other things.

The parser has a dictionary of about 60000 word forms. It has a good coverage of a wide variety of syntactic constructions, including many rare and idiomatic ones. The link grammar dictionary consists of a collection of entries, each of which defines the linking requirements of one or more words. These requirements are specified by means of a formula of connectors combined by the binary associative operators & and or. Precedence is specified by means of parentheses. Without loss of generality we may assume that a connector is simply a character string ending in + or -.

When a link connects to a word, it is associated with one of the connectors of the formula of that word, and it is said to satisfy that connector. No two links may satisfy the same connector. The connectors at opposite ends of a link must have names that match, and the one on the left must end in + and the one on the right must end in -. In basic link grammars, two connectors match if and only if their strings are the same (up to but not including the final + or -). The connectors satisfied by the links must serve to satisfy the whole formula.

A sequence of words is a sentence of the language defined by the grammar if there exists a way to draw links among the words so as to satisfy each words formula, and the following meta-rules:

- Planarity: The links are drawn above the sentence and do not cross.
- Connectivity: The links suffice to connect all the words of the sequence together.
- Ordering: When the connectors of a formula are traversed from left to right, the words to which they connect proceed from near to far. In other words, consider a word, and consider two links connecting that word to words to its left. The link connecting the nearer word (the shorter link) must satisfy a connector appearing to the left (in the formula) of that of the other word. Similarly, a link to the right must satisfy a connector to the left (in the formula) of a longer link to the right.
- Exclusion: No two links may connect the same pair of words.

The parser reads in a dictionary and parses sentences according to the given grammar. The parser does an exhaustive search - it finds every way of parsing the given sequence with the given link grammar. The order of complexity of a link link grammar parser is $O(n^3)$ (where n is the number of words in the sequence to be parsed). The parser also makes use of several very effective data structures and heuristics to speed up parsing.

2.3.2 DG Parser: Pro3Gres

Pro3Gres[129] stands for PRObability-based, PROlog-implemented Parser for RObust Grammatical Relation Extraction System. It is a fast, broad-coverage, deep-syntactic parsing system based on dependency grammar. It uses hybrid models such as hand-written rules and statistical lexicalization information from the Penn Treebank corpus.

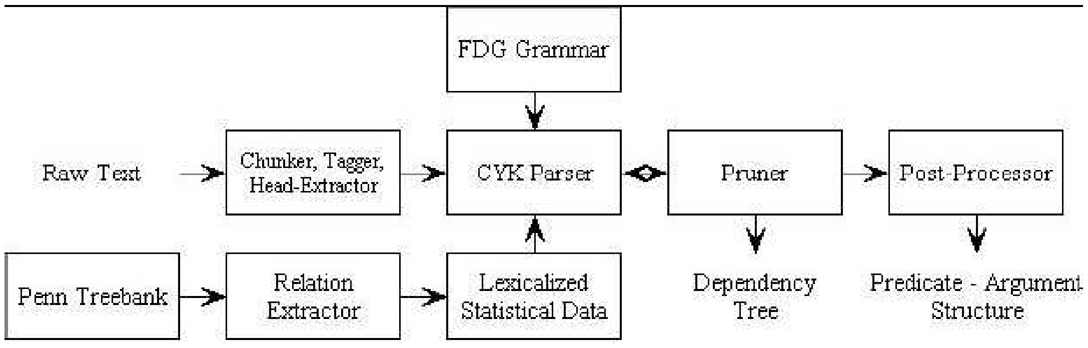


Figure 2.1: Pro3Gres Architecture

The grammar contains around 1000 rules containing the dependents and the heads tag, the direction of the dependency, lexical information for closed class words, and context restrictions. Context restrictions express e.g. that only a verb which has an object in its context is allowed to attach a secondary object. Combining Frank's projection of F-structures from chunks model with statistical techniques a parser that outputs LFG F-structure like structures that is representationally minimalist, combines shallow and deep analysis, is deep-linguistic, robust, fast and psycho-linguistically plausible has been developed. Some post processing techniques have also been proposed to deal with long distance dependencies, reducing the complexity of the parser to the complexity of CFGs.

2.3.3 LFG-DOP parser

Accordance to Bod[13], a particular DOP model is described by specifying settings for the following four parameters: 1) a formal definition of a well-formed representation for utterance analyses, 2) a set of decomposition operations that divide a given utterance analysis into a set of fragments, 3) a set of composition operations by which such fragments may be recombined to derive an analysis of a new utterance, and 4) a probability model that indicates how the probability of a new utterance analysis is computed.

In the parser presented by Bod[14], the fragments for LFG-DOP consist of connected subtrees whose nodes are in F-correspondence with the corresponding sub-units of F-structures. In LFG-DOP the operation for combining fragments, the C-structures are combined by left-most substitution subject to the category-matching condition which is followed by the recursive unification of the F-structures corresponding to the matching nodes. It uses Monte Carlo techniques to compute the most probable parse.

2.3.4 MINIPAR

Dekang Lin [80] proposed a parser called PRINCIPAR based on Government-Binding(GB) theory principles. It contains a lexicon with over 90,000 entries, constructed automatically by applying a set of extraction and conversion rules to entries from machine readable dictionaries. A message passing algorithm is used to construct a shared parse forest according to GB principles. Only preliminary evaluation on some sentences is available.

Dekang Lin [81] also proposed a dependency based evaluation of a parser called MINIPAR. MINIPAR represents the grammar as a network where the nodes represent grammatical categories and the links represent types of syntactic (dependency) relationships. The grammar network consists of 35 nodes and 59 links. Additional nodes and links are created dynamically to represent subcategories of verbs. MINIPAR employs a message passing algorithm that essentially implements distributed chart parsing. Instead of maintaining a single chart, each node in the grammar network maintains a chart containing partially built structures belonging to the grammatical category represented by the node. The grammatical principles are implemented as constraints associated with the nodes and links. The lexicon in MINIPAR is derived from WordNet. With additional proper names, the lexicon contains about 130000 entries (in base forms). The lexical entry of a word lists all possible parts of speech of the word and its sub-categorization frames (if any). The lexical ambiguities are handled by the parser instead of a tagger. Like chart parsers, MINIPAR constructs all possible parses of an input sentence. However, it outputs a single parse tree with the highest ranking. Although the grammar is manually constructed, the selection of the best parse tree is guided be the statistical information obtained by parsing a 1-GB corpus with MINIPAR.

A dependency relationship is an asymmetric binary relationship between a word called head (or governor, parent), and another word called modifier (or dependent, daughter). Dependency grammars represent sentence structures as a set of dependency relationships. Normally the dependency relationships form a tree that connects all the words in a sentence. A word in the sentence may have several modifiers, but each word may modify at most one word. The root of the dependency tree does not modify any word. It is also called the head of the sentence. An evaluation with the SUSANNE corpus shows that MINIPAR achieves about 89% precision and 79% recall with respect to dependency relationships.

2.3.5 HPSG Parser

Enju is a syntactic analyzer for English which is based on Head-driven Phrase Structure Grammar (HPSG) developed by TSUJII Junchi et. al. Enju includes a wide-coverage HPSG grammar and its probabilistic model for unification-based grammars. Unlike conventional parsers using CFGs, the output of the parser is a set of predicate-argument relations. They claim that the outputs would be especially useful for high-level NLP applications including information extraction, automatic summarization, and question answering, where the "meaning" of a sentence plays a central role.

Although Unification-based grammars like HPSG provide precise linguistic structures of sentences, their processing is considered expensive because of the detailed descriptions. Hence they have proposed new model for efficient parsing of unification based grammars.

Given set W of words and set F of feature structures, an HPSG is formulated as a tuple, G =$\langle L, R \rangle$, where

$L = \{l = \langle L, R \rangle | w \epsilon W, F \epsilon F\}$ is a set of lexical entries,

R is a set of schemata, i.e., $r \epsilon R$ is a partial function: $F \times F \longrightarrow F$.

Given a sentence, an HPSG computes a set of feature structures, as a result of parsing. They proposed a probabilistic model of unification-based grammars as a log-linear model or maximum entropy model. The probability of parse result T assigned to given sentence w = $\langle w_1, w_n \rangle$ is

$$p(T|w) = \frac{1}{Z_w} \exp(\Sigma_i \lambda_i f_i(T)) \tag{2.1}$$

$$Z_w = \Sigma_{T'} \exp(\Sigma_i \lambda_i f_i(T')) \tag{2.2}$$

where λ_i is a model parameter, and f_i is a feature function that represents a characteristic of parse tree T. Intuitively, the probability is defined as the normalized product of the weights $\exp(\lambda_i)$ when a characteristic corresponding to f_i appears in parse result T. Model parameters λ_i are estimated using numerical optimization methods so as to maximize the log-likelihood of the training data.

They argue that the existing models of probabilistic parsing, for example, Magerman 95; Collins 96; Collins 97; Charniak 97; Collins 99; Hockenmaier & Steedman etc. decompose the probability of a parse result into probabilities of primitive events with assuming the independence of the probabilities. Such models, however, cannot be applied to deep linguistic analysis, because they often violate the independence assumption and the resulting probabilistic model will be inconsistent.

2.4 Brief Survey of Corpus Based Statistical Approaches for Parsing

Before 1990s, most parsers were based on rules of grammar inferred through linguistic studies. However, these rules are often too rigid to accommodate real-world utterances that are still easily comprehensible by human listeners. Also, many sentences are structurally ambiguous according to grammar rules, but they can be easily disambiguated by human listeners. In these cases, correct analysis may require lexical and distributional knowledge not found in hand-crafted grammar rules. It is also very difficult to hand-craft all the grammar rules of a language with manual effort.

Recently, instead of attempting to encode this knowledge manually, which would be too difficult, researchers have turned to corpus-based statistical techniques, in which lexical and distributional knowledge is gathered from large corpora of real human-generated sentences. In addition to increased accuracy, statistical parsers tend to exhibit greater robustness in dealing with unusual utterances, which would cause a more strictly rule-based parser to fail. They also have the advantage of being easier to build and to customize, because they do not require the work of a language expert to carefully design a grammar and patiently encode a dictionary. Instead, given an appropriate framework, all but the most basic grammar rules can be learned automatically from data, resulting in a huge savings in time and effort, especially if an existing parsing system is being ported to a new language or domain.

Parsers which currently show superior accuracies on freely occurring text are corpus-based statistical parsers, since they automatically learn syntactic and semantic knowledge for parsing from a large corpus of text such as a treebank. Treebanks are manually annotated corpora with syntactic information.

Black et al. [42] introduces history-based parsing, in which decision tree probability models, trained from a treebank, are used to score the different derivations of sentences produced by a hand-written grammar. D. Magerman [83] also train history-based decision tree models from a treebank for use in a parser, but do not require an explicit hand-written grammar. Several other recent papers use statistics of pairs of head words in conjunction with chart parsing techniques to achieve high accuracy. A head word of a constituent, informally, is the one word that best represents the meaning of the constituent. Parsers vary greatly on how head word information is used to disambiguate possible parses for an input sentence. The parsers in Collins 96 [30], Collins 97 [87] use chart-parsing techniques and head word bigram statistics derived from a treebank. Charniak 97 [22] uses head word bigram statistics with a probabilistic context free grammar, while Goodman 97 [48] uses head word bigram statistics with a probabilistic feature grammar. In this section, we describe some of the full parsing systems which have shown superior accuracies.

2.4.1 SPATTER

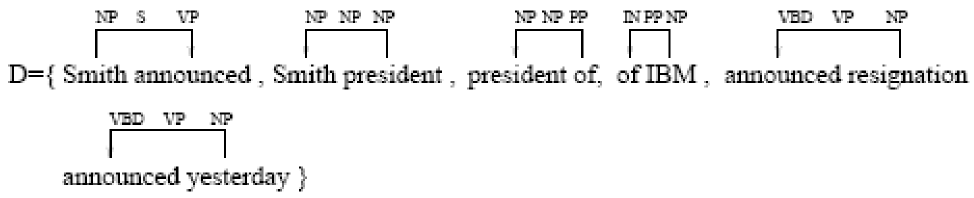
In 1995, Magerman[83], proposed a parser called Spatter (Statistical PATTErn Recognizer), based on a statistical decision-tree learning model combined with a bottom-up parser. In the learning phase the decision-tree classification algorithms identify features which are relevant for each decision, decide which choice to select based on the values of the relevant features and assign a probability distribution to possible choices. For this it makes heavy use of lexical information and evaluates POS tags of previous encountered words. Each node in the tree carries four pieces of information: the headword (head's terminal), the head's part-of-speech tag (head's pre-terminal), the syntactic category (if not pre-terminal) and a category describing its relationship (root, different types of children) to other dependents. Spatter was the first parser which was trained and tested on the Wall Street Journal section of the Penn Treebank.

2.4.2 Collins Parser

Michael Collins[30], in 1996, proposed a statistical parser based on probabilities of dependencies between head-words in the parse tree. The parser uses simple bottom-up chart parser to produce parse trees. The dependency model maps the trees to dependency structures. There are two components in his parser. First, the statistical component assigns a probability to every candidate parse tree for a given sentence i.e. given a sentence S and a tree T, the model estimates the conditional probability $P(T|S)$. The tree can be further represented as a set of base NPs[118] B, and a set of dependencies D.

Example: John Smith, the president of IBM, announced his resignation yesterday.

B = [John Smith] [the president] [IBM] [his resignation] [yesterday]


NP S VP
NP NP NP
NP NP PP
IN PP NP
VBD VP NP
D={ Smith announced , Smith president , president of, of IBM , announced resignation
VBD VP NP
announced yesterday }

$P(T|S) = P(B, D|S) = P(B|S)P(D|S, B)$

The second component finds T_{best}.

The parser was trained on sections 02-21 of Wall Street Journal portion of the Penn Tree-bank corpus. This contains approximately 40,000 sentences. The section 23 which contains 2416 sentences was used for testing. The performance was measured in terms of labelled precision and recall.

Michael Collins[87], in 1997, extended his parser by three new parsing models. Model 1 is essentially a generative version of the model described in Collins 96. In Model 2, he extended the parser to make the complement/adjunct distinction by adding probabilities over sub-categorization frames for head-words. In Model 3 he gave a probabilistic treatment of wh-movement, which is derived from the analysis given in Generalized Phrase Structure Grammar. In this parser, he used lexicalized context-free grammar. A PCFG can be lexicalised by associating a headword with each non-terminal in a parse tree. Parsing accuracy is defined as the ratio of correct dependency links vs. the total number of dependency links in a sentence.

Michael Collins[29] did some modifications in 1999 to his parser to apply for Czech language which is a relatively free word order language. The modifications such as punctuations as phrase boundaries, a way to deal with co-ordination and relative clauses, bigram modal for dependencies and tag set reduction were made. He then tested this parser for English language and achieved 91% accuracy.

2.4.3 Charniak Parser

Charniak[22] in 1997 proposed a system that induces grammar and probabilities from a hand-parsed corpus. It produces the parse π of a sentence S that maximizes the given model $P(\pi|S)$. The model can be written as

$$P(S) = \arg\max_{\pi} \frac{P(\pi, S)}{P(S)} = \arg\max_{\pi} P(\pi, S) \tag{2.3}$$

The parser assigns the probabilities $P(\pi, S)$ to the sentence s under all its possible parses π and then chooses the best parse for which $P(\pi|S)$ is highest. Two assumptions are made while evaluating the model:

1. The probability of head is dependent only on its type `t`, the type of the parent constituent `l` and the head of the parent constituent `h`. Thus we use $P(s|h, t, l)$.

2. The probability of the form of the constituent given its head i.e. the probability that a constituent `c` is expanded given a grammar rule given that `c` is type of `t`, is headed by `h`, and has a parent of type `l` i.e $P(r|h, t, l)$.

Charniak evaluated his parser in three ways: 1) with only PCFG rules 2) with the additional two assumptions $P(s|h,t,l)$, $P(r|h,t,l)$ 3) basic model plus the statistics based on unsupervised learning on 30 Million words of Wall Street Journal text.

In 2000, he proposed a new model based on maximum entropy models[23]. The model assigns the probability in a top down process considering each constituent c in π and for each c guessing the pre-terminal t(c), then the lexical head of c i.e. h(c), and then expansion of c into further constituents e(c). The probability of the parse can be written as

$$P(\pi) = \Pi_c \epsilon \pi p(t|l,H).p(h|t,l,H).p(e|l,t,h,H). \tag{2.4}$$

The conditional probabilities are computed using log-liner models.

2.4.4 Stanford Lexicalized Parser

Dan Klein and Christopher D. Manning [69, 34] proposed in 2002 Fast Exact Inference with a Factored Model for Natural Language Parsing. Stanford Lexicalized parser contains implementation of a factored product-of-experts model, with separate PCFG phrase structure and lexical dependency experts, whose preferences are combined by efficient exact inference, using an A* algorithm. The parser can also be used as an accurate unlexicalized stochastic context-free grammar parser.

Generative models for parsing typically model one of the kinds of structures given below:

1) a plain phrase-structure tree T , which primarily models syntactic units,

2) a dependency tree D, which primarily models word-to-word selectional affinities,

3) a lexicalized phrase-structure tree L, which carries both category and (part-of-speech tagged) head word information at each node.

A lexicalized tree can be viewed as the pair L = (T,D) of a phrase structure tree T and a dependency tree D. In this view, generative models over lexicalized trees can be regarded as assigning mass P(T,D) to such pairs. The developers factored their model as P(T,D) = P(T).P(D).

For P(T), the developers used PCFGs. The simplest, PCFG-BASIC model, uses the raw treebank grammar, with non-terminals and rewrites taken directly from the training trees. For improved models of P(T), tree nodes labels were annotated with various contextual markers. In PCFG-PA model, each node was marked with its parent's label. They say that it improves the accuracy of PCFG parsing by weakening the PCFG independence assumptions. The best PCFG model, PCFG-LING, involved selective parent splitting, order-2 rule Markovization, and linguistically-derived feature splits.

Models of P(D) were lexical dependency models, which deal with tagged words: pairs $\langle w, t \rangle$. First the head $\langle w_h, t_h \rangle$ of a constituent is generated, then successive right dependents $\langle w_d, t_d \rangle$ until a STOP token is generated, then successive left dependents until STOP token is generated again.

The core of their parsing algorithm is a tabular agenda-based parser. An agenda-based parser tracks all edges that have been constructed at a given time. When an edge is first constructed, it is put on an agenda, which is a priority queue indexed by some score for that node. The agenda is a holding area for edges which have been built in at least one way, but which have not yet been used in the construction of other edges. The core cycle of the parser is to remove the highest-priority edge from the agenda, and act on it according to the edge combination schema, combining it with any previously removed, compatible edges.

Here we give a comparative feel for the performance of various parsers:

Table 2.1: Labelled Precision(P), Labelled Recall(R) of Recent Full parsing Systems

	< 40 words		< 100 words	
Parser	LP in %	LR in %	LP in %	LR in %
Magerman (1995)	84.9	84.6	84.3	84.0
Collins (1996)	86.3	85.8	85.7	85.3
Charniak (1997)	87.4	87.5	86.6	86.7
Apple Pie Parser	77.18	79.55		
Collins (1999)	88.7	88.5	88.3	88.1
Charniak (2000)	90.1	90.1	89.5	89.6

2.5 Difficulties with Full Parsing Systems

During the past decade or so a number of new grammar formalisms have been introduced for a variety of reasons, for example, eliminating transformations in a grammar, accounting linguistic structures beyond the reach of context-free grammars, integrating syntax and semantics directly, etc. But none of them became successful in solving the problems in parsing because of the following reasons:

1. A disadvantage with grammar-based parsing systems is that natural language often does not conform to the rules of the grammar. Unusual constructions, casual speech, innovative expressions, mistakes, noise, and interruptions can all result in sentences that are quite understandable to a human reader or listener, but utterly confusing to a rule-based parser.

2. It is very difficult to write a complete and tight grammar (where tight means that it does not produce lots of incorrect analyses). Beyond the core grammar generally discussed by linguists, there are a large number of relatively rare constructs. If we simply add productions for all these rare constructs, they end up 'firing' by producing lots of bad parses. This leads us to write ever-more-complex constraints on the grammar.

3. Developing wide coverage grammars has proved to be a very difficult task. The best available grammars, even for languages such as English, are far from perfect today. When it comes to Indian languages, although so much

is talked about in theory, there is no substantive computational grammar for any of our languages.

4. Good hand-written grammars are quite large (at least several hundred productions) and complex (many features and grammatical constraints).

5. To compound the problem, even if we are successful in parsing, we may get a very large number of parses for a single sentence if we rely on grammatical constraints alone. There would be no easy way to find which of these is really the correct parse.

6. Even if we want to use Machine Learning Techniques, the training of a full parser requires a large collection of fully parsed sentences as a training corpus which is a rare resource for most languages. Even for languages like English, there are not many parsed corpora to choose from.

7. Even if there exist parsed corpora, finding a parsed corpus suitable to our grammar is not trivial. Either we have to do some approximations or we have to consider the same grammar in which corpus is available for building the systems.

8. For many NLP applications where there is no need for full text understanding, it is sufficient to use shallow parsing. Hence the interest in shallow or partial parsing.

2.6 Shallow Parsing

Gee and Grosjean [45, 4] gave in 1983 psychological evidence for the existence of chunks in terms of performance structures. Performance structures are the structures of word clustering that emerge from a variety of types of experimental data, such as pause durations in reading, and naive sentence diagramming. Gee and Grosjean showed that performance structures are best predicted by ϕ-phrases. ϕ-phrases can be obtained by breaking the input string after each syntactic head that is a content word (with the exception that function words syntactically associated with a preceding content word group with the preceding content word). The chunks of sentence are ϕ-phrases.

Steve Abney [4], in 1991, modified the definition of chunk which is the basis for all recent chunking tasks. Steve Abney defined chunk as “a syntactic structure, which comprises a connected subgraph of the sentence's global parse-tree, and contains a major head". Major heads are all content words except those that appear between a function word f and the content word that f selects. Let h be a major head. The root of the chunk headed by h is the highest node in the parse tree for which h is the s-head, that is, the semantic head. Intuitively, the s-head of a phrase is the most prominent word in the phrase. If the syntactic head h of a phrase P is a content word, h is also the s-head of P. If h is a function word, the s-head of P is the s-head of the phrase selected by h.

The concept of partial or shallow parsing [2] was also introduced by Steven Abney in 1994. Partial Parsing means "the task of recovering only a limited amount of (domain specific) syntactic information from natural language sentences". Shallow or partial parsing involves several different tasks, such as text chunking, noun phrase chunking or clause identification. Text chunking consists of dividing an input text into non-overlapping segments. These segments are non-recursive. Noun phrase chunking (NP chunking) is a part of the text chunking task, which consists of detecting only noun phrase chunks. The aim of the clause identification task is to detect the start and the end boundaries of each clause (sequence of words that contains a subject and a predicate) in a sentence.

In CoNLL-2000 chunking task [124], chunking was defined as "the task of dividing a text into phrases in such a way that syntactically related words become members of the same phrase". Here the sentence is divided into non-overlapping phrases.

For example, the sentence "*He reckons the current account deficit will narrow to only # 1.8 billion in September.*" can be divided as follows:

```
[NP He ] [VP reckons ] [NP the current account deficit ] [VP will
narrow ] [PP to ] [NP only # 1.8 billion ] [PP in ] [NP September ].
```

Shallow parsing has become an interesting alternative to full parsing. Although the detailed information from a full parse is lost, shallow parsing can be done on non-restricted texts in an efficient and reliable way. In addition, partial syntactic information can help to solve many natural language processing tasks. For example, identifying named entities found useful in applications such as information retrieval, information extraction and text summarization. One more example is within the Verbmobil project - shallow parsers have been used to add robustness to a large speech-to-speech translation system [146].

X. Li and D. Roth [77] gave in 2001 evidence for the argument that shallow parser is more robust and reliable than full parser by comparing two state-of-the-art parsers. They also tried to prove incremental and modular parsing may result in more robust parsing than directly going for full parsing. They finally observed that shallow parsers that are specifically trained for identifying phrases are more accurate and robust than full parsers. They have used Collins parser and SNoW based shallow parser for this purpose.

The full parser used for this purpose was developed by Michael Collins. Michael Collins [30] proposed in 1996 a statistical parser based on probabilities of dependencies between head-words in the parse tree. The parser uses simple bottom-up chart parser to produce parse trees. The dependency model maps the trees to dependency structures. The Shallow Parser used is SNoW based CSCL parser developed by Andrew J. Carlson et. al.[17]. The SNoW (Sparse Network of Winnows) learning architecture is a multi-class classifier that is specifically tailored for large scale learning tasks and for domains in which the potential number of features taking part in decisions is very large, but may be unknown a priori. It learns a sparse network of linear functions in which the target concepts (class labels) are represented as linear functions over a common feature space.

The full parser yielded an F-measure of 91.96% for detecting 11 types of phrases in text whereas the shallow parser yielded 94.64%.

2.6.1 Shallow Parsing Systems: An Overview

In the year 1995, Ramshaw and Marcus [118] proposed transformation-based learning to identify chunks in texts by treating chunking as a tagging problem. The chunk structure was represented as tags attached to words, in a similar way as is done in data-driven POS tagging. They performed experiments using two different chunk structure targets. The first experiment was to identify non-overlapping, non-recursive noun phrases, so called base NPs. They contain the nominal head, including determiners and adjectives, but not prepositional phrases or post nominal modifiers after head noun. The second experiment was to partition sentences into non-overlapping noun-type (N) and verb-type (V) chunks. The noun-type chunks consists of noun phrases with the nominal head, prepositional phrases including an NP argument, but not coordinating conjoined NPs. They have used Wall Street Journal articles from Penn Treebank as training data. They obtained a precision of 91.8% and a recall of 92.3% for base np chunks when trained on 200000 words using lexical and POS information. When lexical information was excluded, precision and recall decreased to 90.5% and 90.7% respectively. In the second experiment, they obtained precision and recall of 87.7% and 88.5% respectively when training was performed on 200000. Also, they pointed out that the size of the training set has a significant effect on the results.

Brants [15] presented in 1999 a method for partial parsing that uses cascades of Markov Models. Instead of a single word or a single symbol, each state of the proposed Markov Models emits context-free partial parse trees. Each layer of the resulting structure is represented by its own Markov Model, hence the name Cascaded Markov Models. The output of each layer of the cascades is a probability distribution over possible bracketings and labelings for that layer. This output forms a lattice and is passed as input to the next layer. Number of layers will decide number of phrases that can be parsed by the model. The algorithm generates the internal structure of np and pp chunks including ap, advp and other pre-modifiers. vp and co-ordination were excluded in this work. The algorithm was tested on 300000 words taken from the NEGRA corpus consisting of German newspaper texts. Recall was 54% for 1 layer and 84.8% for 9 layers; precision was 91.4% for 1 layer and 88.3% for 9 layers.

Roberto Basili et. al. [9] developed in 1999 an architecture for parsing called the Chaos architecture. The basis is that verbs determine the semantics of a sentence and its surface realization is strictly dependent on this fact. Verbs characterize the set of syntactic restrictions over the grammatical representation of the target sentence. The first stage identifies phrases i.e. the cores of nominal phrases, prepositional phrases, adjectival phrases, and verbal phrases. The second stage uses the verb sub-categorization lexicon in order to detect the verb arguments in the sentence. The adopted strategy investigates the arguments of verbs exploiting the approximation of clause boundaries. Chunks are used as input to the Clause Boundary Recognition (CBR) aiming to recognize clauses and their hierarchical relations. The recognition of clauses is integrated with a Verb Shallow Recognizer(VSG) to detect relations between a verb and its arguments. The interaction between the CBR and VSG provides a combined recognition of the clause hierarchy and the set of argumental dependencies of verbs, namely Verbal inter-chunk dependencies(Vicds). Finally, the third step of analysis is the Shallow recognizer (SG) triggered by Chunks, the clause hierarchy H and the detected argumental relations. The final representation of the sentence is a graph whose nodes are words and whose edges are inter-chunks dependencies. F-measures for V-Sub, V-obj and V-pp relations were 82%, 78% and 77% respectively.

Walter Daelemans et al. [32] proposed in 1999 a memory based learning approach to shallow parsing. Memory based learning is a classification based, supervised learning approach. It constructs a classifier for a task by storing set of examples each associated with a feature vector with one of the finite number of classes. Given a new feature vector, the classifier extrapolates its class from most similar feature vectors in memory. One of the distance metric they used IB1-IG. In this the distance between test and memory items is defined as the number of features for which there is a different value. Since in most cases not all features are relevant for the task, the algorithm uses information gain to weight the cost of a feature value mismatch during the comparison. The developers have got an F-measure of 93.8% for NP chunking, 94.7% for VP chunking, 77.1% for subject detection and 79.0% for object detection.

In CoNLL 2000 task [124], they have chosen Wall Street Journal(WSJ) corpus which is a part of the Penn TreeBank II corpus for training and testing. The Penn Treebank Project Release 2, 1995 version consists the new Penn Treebank II bracketing style, which is designed to allow the extraction of simple predicate/argument structure. Over one million words of 1989 Wall Street Journal material have been annotated in Treebank II style.

The training corpus was obtained from the sections 15-18 of Wall Street Journal corpus (WSJ). The training data consists of 211727 tokens and 106978 chunks. Section 20 of WSJ corpus was used as test data. The test data contains 47377 tokens and 24002 chunks. The annotation of the data has been derived from the WSJ corpus by a program written by Sabine Buchholz from Tilburg

University, The Netherlands. The baseline result was obtained by selecting the chunk tag which was most frequently associated with the current part-of-speech tag.

The training and test data consist of three columns separated by spaces. Each word has been put on a separate line and there is an empty line after each sentence. The first column contains the current word, the second its part-of-speech tag as derived by the Brill tagger and the third its chunk tag as derived from the WSJ corpus. The chunk tags contain the name of the chunk type, for example I-NP for noun phrase words and I-VP for verb phrase words. Most chunk types have two types of chunk tags, B-CHUNK for the first word of the chunk and I-CHUNK for each other word in the chunk. The O chunk tag is used for tokens which are not part of any chunk. Instead of using the part-of-speech tags of the WSJ corpus, the data sets used tags generated by the Brill tagger.

While extracting chunks from the WSJ corpus, they made some simplifications. They considered that a chunk can contain only pre-modifiers of the head but no post-modifiers and arguments. So they consider preposition phrase contains only preposition not an argument NP and SBAR chunk contains only complementizer.

Eleven types of phrases that are considered for evaluating the performance are given below:

1) Noun Phrase(NP), 2) Verb Phrase(VP), 3) Prepositional Phrase(PP), 4) Adverb Phrase(ADVP), 5) Adjective Phrase(ADJP), 6) Subordinated Clause(SBAR), 7) Conjunction Phrase(CONJP), 8) List Markers(LST), 9) Interjections(INTJP), 10) Particles(PART), 11) Unlike Coordinated Phrases(UCP)

Eleven systems participated in the CoNLL-2000 shared task [46] on chunking. In the shared task, all 11 systems outperformed the baseline. Two systems performed a lot better: Support Vector Machines used by Kudoh and Matsumoto and Weighted Probability Distribution Voting used by Van Halteren.

- Taku Kudoh and Yuji Matsumoto [74] used SVMs for chunking in CoNLL chunking task. This system performed the best in the CoNLL 2000 chunking task. They considered chunking as a classification problem. They used pairwise SVMs for classification task. They have used IOB model to label chunks. They used the contexts of words, their POS tags and chunk labels as features. They used a window of 5 for words and POS tags and for chunk labels window of 2.

- RoB Koeling [71] used maximum entropy models for chunking. He used features such as current word, POS tag of current word, surrounding words and POS tags of surrounding words.
- Christer Johansson [59] used context sensitive maximum likelihood approach for chunking task. He used only POS tag information to find the chunk label. He also verified that using larger contexts than 5 does not show any significant improvement.
- Erik F. Tjong Kim Sang [125] used combinations of memory based learning systems for chunking task. He used a variety of bracketing structures including IOB1, IOB2, IOE1 and IOE2. He also used one more information - whether the words can start the chunk and whether the words can end the chunk or not. Finally he used voting methods to select the best chunk from different classifiers.
- Guo Dong Zhu, Jain Su and Tam Guan Tey [154] used error driven HMM based Chunk tagger with context dependent lexicon. He combined memory based learning algorithm to this.
- Pla and Molina [110] used language models expressed in terms of finite state transducers for chunking task.
- Herve Dejean [36] used ALLiS (Architecture for learning linguistic structure) which is symbolic machine learning system for chunking task.
- Miles Osborne [103] used Maximum entropy based POS tagger for chunking Task.
- Jorn Veenstra and Antal Van Den Bosch [144] used single classifier Memory based phrase chunking.
- Marc Vilain and David Day [145] used rule based sequence processors for chunking task.
- Hans Van Halteren [143] used Weighted probability distribution voting algorithm(WPDV) for chunking Task.

Table 2.2: Performance of the eleven systems in the CoNLL-2000 chunking task

Parsing System	Precision	Recall	F-measure
Kudoh and Matsumoto	93.45	93.51	93.48
Van Halteren	93.13	93.51	93.32
Tjong Kim Sang	94.04	91	92.5
Zhou Tey and Su	91.99	92.25	92.12
Dejean	91.87	92.31	92.09
Koeling	92.08	91.86	91.97
Osborne	91.65	92.23	91.94
Veenstra and Van Den Bosch	91.05	92.03	91.54
Pla Moliena and Prieto	90.63	89.65	90.14
Johanson	86.24	88.25	87.23
Vilain and Day	88.82	82.91	85.76
Base line	72.58	82.14	77.07

Tjong Kim Sang [126] considered in 2002 issues in applying memory-based learning (MBL) to shallow parsing. In his paper, a weakness of MBL, namely that it can have difficulty handling large numbers of features was identified. A feature selection method, namely bidirectional hill climbing was found to yield insignificant gains in performance for NP parsing. However, it did produce a significant improvement for clause identification. Tjong Kim Sang also showed how ensemble learning techniques such as (weighted) majority voting and stacking could improve upon performance. All system combination methods improved on the results of the individual MBL classifiers, and the best performer was to employ MBL itself as a stacked classifier.

Zhang et al. [153] presented in 2002 a generalized version of the Winnow algorithm. They observed that the original Winnow algorithm is only guaranteed to converge on linearly separable data. So, given the possibility that features for shallow parsing are not linearly separable, the authors modified Winnow such that it would converge, even for non-linearly separable features. They also showed that both versions of Winnow were robust against irrelevant features. The system performed better than the best system in CoNLL chunking task.

Megyesi et al. [85] used POS taggers for shallow parsing in 2002. The shallow parsers are based on three state-of-the-art data-driven algorithms that have implementations for the POS tagging approach. The taggers that are used to parse Swedish in this study are: fntbl which is a fast version of Brill's tagger based on transformation based learning, mxpost, based on the maximum entropy framework and TrigramsnTags (tnt) based on a Hidden Markov Model. Unlike the others, this system dealt with shallow parsing for Swedish, and not English. Unlike other studies, the author found that ignoring lexical information improved performance of all of her systems.

Dejean et al. [37] presented in 2002 a top-down rule induction system, called ALLiS, for learning linguistic structures. The initial system was enhanced with additional mechanisms to deal with noisy data. The author identifies two types of difficulties: significant noise in the data and the presence of linguistically motivated exceptions. To address these problems, a refinement algorithm is introduced to learn exceptions for each rule that is learned. The second improvement introduces linguistically motivated prior knowledge to improve the efficiency and accuracy of the system. The experimental results clearly demonstrated significant improvement with the introduction of the two mechanisms. In comparison to Brill's a well-known transformation based learning system (TBL), ALLiS needs fewer rules and overcomes a number of classification errors produced by TBL.

Osborne et al. [104], in 2002, considered an issue, namely what happens when the training set is either noisy, or else drawn from a different distribution from the testing material. This paper took a range of shallow parsers (including both single model parsers and ensemble parsers) and trained them using various types of artificially noisy material. In a second set of experiments, the issue of whether naturally occurring disfluencies have more impact on performance than a change in the distribution of the training material was investigated. It was found that the changes in the distribution are more important. The author drew various conclusions from this work. Shallow parsers are robust and only large quantities of noise will significantly impair performance. No single technique worked best with all types of noise with different kinds of noise favouring different parsers. Regarding the results on changes in the distribution of training data, the clear lesson is that if one wishes to improve the performance of shallow parsers on a particular task, it is better to annotate more examples from the target distribution than to use additional training material from other distributions. One surprise in this paper is that the parsers employing system combination, although generally the best performers in the literature, were not always the best at dealing with noise.

Fei Sha and Fernando Pereira [131] used in 2003 conditional random fields for shallow parsing by considering it as a sequence labelling task. They showed that Conditional random fields for sequence labeling offer advantages over both generative models like HMMs and classifiers applied at each sequence position. Conditional random fields (CRFs) [147] are a probabilistic framework for labeling and segmenting sequential data. Based on the conditional approach. CRFs define conditional probability distributions $p(Y|X)$ of label sequences given input sequences. The primary advantage of CRFs over Hidden Markov Models is their conditional nature, resulting in the relaxation of the independence assumptions required by HMMs in order to ensure tractable inference. Additionally, CRFs avoid the label bias problem, a weakness exhibited by maximum entropy Markov models (MEMMs) and other conditional Markov models based on directed graphical models.

Libin Shen and Aravind K. Joshi [133] proposed a SNoW based Supertagger for NP Chunking. Supertagging is the tagging process of assigning the correct elementary tree of LTAG (lexical Tree Adjoining Grammar), or the correct supertag, to each word of an input sentence. Let $W = w_1, w_2...w_n$ be the sentence, $Q = q_1, q_2...q_n$ be the POS tags, and $T = t_1, t_2...t_n$ be the supertags respectively. For each POS tag q, they construct a SNoW classifier to estimate the distribution $P_q(t|t', W, Q)$ within a 5-word window plus two head supertags before the current word. They achieved a high performance for NP chunking. An F-measure of 95.18% is obtained on WSJ corpus-section 20.

Table 2.3: Performance of recent chunkers

Parsing System	Precision	Recall	F-measure
Tjong Kim Sang	94.04	91.00	92.50
Zhang , Damerau , David Johnson	94.28	94.07	94.17
Mufioz, Punyakanok, Roth, Zimak	-	-	92.0
Fei Sha and Fernando Pereira	-	-	94.38

2.7 Clause Identification

In CoNLL clause identification task [127], clauses are defined as "*word sequences which contain a subject and a predicate*". Clause identification is the next step towards full parsing. Clauses can be embedded in each other, so this is more difficult than chunking.

```
(S The deregulation of railroads and trucking companies
        (SBAR that
                (S began in 1980)
        )
        enabled
        (S shippers to bargain for transportation).
)
```

2.7.1 Literature Survey

Papageorgiou [106], in 1997, used a set of handcrafted rules for identifying clause boundaries in text. The system was used as a pre-processing step for bilingual alignment of parallel texts. The system was designed for unrestricted English texts and its results stand at about 93%, when evaluated on 562 clauses contained in the CELEX database.

Constantin Orasan [102] proposed a hybrid method in 2000 for clause splitting in unrestricted English texts. K-nearest neighbour algorithm is used to identify the clause boundaries in the first step. The results of a machine learning algorithm, trained on an annotated corpus, are processed by a shallow rule-based module in order to improve the accuracy of the method. The evaluation of the results showed that the machine learning algorithm is useful for identification of clauses boundaries and the rule-based module improves the results. Firstly he evaluated the results of the machine learning algorithm. From a total of 28270 clause boundaries the algorithm correctly identified 23348 and produced 4922 under-recognition and 2202 over-recognition errors. This results in high recall, 91.38%, but poor precision 82.58%. In the next step, he evaluated the results after applying the rules. In this case the number of correctly identified clauses went up to 24986, but the number of over-recognition errors also rose to 3755, resulting 86.84% recall and 89.01% precision.

In the CoNLL-2001 shared task [127], the goal was to identify clauses in sentences. The training and test data are from Wall Street Journal corpus (WSJ): sections 15-18 as training data (211727 tokens) and section 20 as test data (47377 tokens). The annotation of the data has been derived from the WSJ corpus by a program written by Sabine Buchholz from Tilburg University, The Netherlands. The baseline results were produced by a system that assume every sentence contain one clause which contains the complete sentence.

In this shared task, the goal was find clauses in text. Since this task is a little difficult, they disregarded type and function information of clauses i.e every clause is tagged as S rather than using more elaborate tags. The shared task was divided into three parts: 1) identifying clause start 2) recognizing clause end and 3) finding complete clauses.

They have used WSJ sections of 15 to 18 of Penn Tree Bank as training material, section 20 as development material for tuning the parameter of the learner and section 21 as test data. The data sets contain tokens, information about location of sentence boundaries and information about clause boundaries. For all three parts of the shared task, the clause segmentation methods were evaluated with the F rate, which is a combination of the precision and recall rates:

F = 2*precision*recall / (recall+precision)

The following are the systems that participated in CoNLL-2001 clause identification task:

- Carreras and Marquez [20] used Ada-Boost learning algorithm for clause identification. This algorithm is useful for obtaining highly accurate classification rule by combining many weak classifiers, each of which may be only moderately accurate. The boosting process involves two steps:

  1. Instead of random sample of a training data, used weighted samples to focus learning on most complex examples
  2. Instead of combining classifiers with equal vote, used a weighted vote.

- Antonio Molina and Ferran Pla [90] used HMM based approach for clause identification. They used specialized HMMs called lexicalized HMMs for this task. They interpreted the three parts of shared task as tagging problems and used HMMs to find out most probable sequence of tags given an input sequence. In the third part they used rules for fixing the bracketing problem.

- Herve Dejean [35] used ALLiS (Architecture for learning Linguistic structure) - a symbolic machine learning system for clause identification. It is based on theory refinement, which means that it adapts grammars. The learner select a set of rules based on their prediction accuracy in the training corpus.

- Erik F. Tjong Kim Sang [128] used memory based learning technique for the clause identification. This has given good results for identifying one of the clause boundaries but not for identifying full clauses. He used heuristics for converting part 1 and part 2 results into results of part 3.

- James Hammerton [54] used a feed-forward neural network based architecture, long-short term memory (LSTM), for predicting embedded clause structures. The network processes sentences word by word. Memory cells in its hidden layer enable it to remember its states with information about current clause.

- Jon D Patrick and Ishaan Goyal [109] used boosted decision graphs based on Ada-Boost learning algorithm for clause identification.

Table 2.4: Performance of various systems in CoNLL-2001 task: recognizing complete clauses

system	P	R	F
Carreras and Mquez	84.82	73.28	78.63
Molina and Pla	70.89	65.57	68.12
Tjong Kim Song	76.91	60.61	67.79
Patrik and Goyal	73.75	60.00	66.17
Dejean	72.56	54.55	62.77
Hammerton	55.81	45.99	50.42
Baseline	98.44	31.48	47.71
Carreras et. al.[2002]	90.18	72.59	80.44

Georgiana Puscasu [115] proposed in 2004 a multilingual method for detecting clause boundaries in unrestricted texts. The method combines language independent machine learning techniques like memory based learning algorithm with language specific rules in order to take the first step in building the hierarchical structure of sentences. The results of a machine learning algorithm, trained on an annotated corpus, are processed by a rule-based module which deals with clause boundaries which is not included in the learning process. Formal indicators of coordination and subordination, together with verb type information (finite or non-finite) are used for identifying clause boundaries. The method was evaluated on Romanian and English and the F-measure for clause start detection was 95.30% for Romanian and 92.38% for English. The F-measure obtained for identifying complete clauses was 88.76% for Romanian and 82.36% for English.

2.8 Shallow Semantic Parsing

The shared task of CoNLL-2004 [18] concerned with the recognition of semantic roles of different constituents in a given sentence for the English language. They referred to it as Semantic Role Labeling (SRL). Given a sentence, the task consists of analyzing the propositions expressed by some target verbs of the sentence. In particular, for each target verb all the constituents in the sentence which fill a semantic role of the verb have to be extracted. Typical semantic arguments include Agent, Patient, Instrument, etc. and also adjuncts such as Locative, Temporal, Manner, Cause, etc. The challenge for CoNLL-2004 shared task was to come up with machine learning strategies which address the SRL problem on the basis of only partial syntactic information, avoiding the use of full parsers and external lexico-semantic knowledge bases. The annotations provided for the development of systems include, apart from the argument boundaries and role labels, words, POS tags, base chunks, clauses, and named entities.

They used Proposition Bank (PropBank) which is an annotated corpus of the Penn Treebank with verb argument structure. The semantic roles covered by PropBank are the following:

1) Arguments defining verb-specific roles like agent, patient or theme etc. Their semantics depends on the verb and the verb usage in a sentence, or verb sense.

2) Adjuncts (AM-): General arguments that any verb may take optionally.

There are 13 types of adjuncts: 1) AM-ADV : general-purpose 2)AM-MOD : modal verb 3)AM-CAU : cause 4)AM-NEG : negation marker 5)AM-DIR : direction 6)AM-PNC : purpose 7)AM-DIS : discourse marker 8)AM-PRD : predication 9)AM-EXT : extent 10)AM-REC : reciprocal 11)AM-LOC : location 12)AM-TMP : temporal 13)AM-MNR : manner

3) References (R-): Arguments representing arguments realized in other parts of the sentence. The role of a reference is the same as the role of the referenced argument. The label is an R- tag prefixed to the label of the referent, e.g. R-A1.

4) Verbs (V): Participant realizing the verb of the proposition, with exactly one verb for each one.

The following sentence, taken from the PropBank corpus, exemplifies the annotation of semantic roles:

[A0 He] [AM-MOD would] [AM-NEG n't] [V accept] [A1 anything of value] from [A2 those he was writing about] .

Here, the roles defined in the above example are : 1) V: verb 2) A0: acceptor 3) A1: thing accepted 4) A2: accepted-from 5) AM-MOD: modal 6) AM-NEG: negation

The following are the systems that participated in CoNLL-2004 Semantic Role Labeling (SRL) task:

- Antal van den Bosch et al. [142] used memory-based learning approach for the SRL task. Apart from the provided words and the predicted POS tags, chunk labels, clause labels, and named-entity labels provided beforehand, they have considered an additional set of automatically derived features: attenuated words,the distance between the candidate role word and the verb, preceding preposition, passive main verb, current clause, role pattern. They have not employed the Propbank information and verb sense information.

- Xavier Carreras et al. [19] used a two-layer learning architecture to recognize arguments in a sentence and predict the role they play in the propositions. The exploration strategy visits possible arguments bottom-up, navigating through the clause hierarchy. The final architecture makes use of

Voted Perceptrons which compute a prediction as an average of all vectors generated during training.

- Kadri Hacioglu et al. [52] used support vector machines for the SRL task. They have used some phrase level features like token position, path, clause bracket patterns, clause position, head word suffixes, distance and length. The sentence level features are: predicate POS tag, predicate frequency, predicate BP context, predicate POS context, predicate argument frame and number of predicates.

- Derrick Higgins et al. [55] used transformation-based learning (TBL) to the problem of semantic role labeling. They have used 12 features and nearly 120 rules for this task.

- Beata Kouchnir et al. [73] used memory based learning approach TiMBL for this task.

- Joon-Ho Lim et al. [78] used maximum entropy based model for SRL task. For applying the maximum entropy model to semantic role labeling, they took an incremental approach as follows. Firstly, the semantic roles are assigned to the arguments in the immediate clause including a predicate, and then, the semantic roles are assigned to the arguments in the upper clauses by using previously assigned labels. They used nearly 30 features for this task.

- Kyung-Mi Park et al. [107] used support vector machines for the SRl task. They used 29 features for argument identification phase and 27 for role labelling.

- Vasin Punyakanok et al. [114] used SNoW learning architecture for SRL task. Specifically, they used two classifiers, one to detect beginning phrase locations and a second to detect end phrase locations. They have used 10 features for this task. The second task is accomplished in two steps. First, a multi-class classifier is used to supply confidence scores corresponding to how likely individual phrases are to have specific argument types. Then they look for the most likely solution over the whole sentence, given the matrix of confidences and linguistic information that serves as a set of global constraints over the solution space.

- Ken Williams et al.[149] used transformation based learning approach for SRL task.
- Ulrike Baldewein et al.[7] used a Maximum Entropy learner, augmented by EM-based clustering to model the fit between a verb and its argument candidate.

Table 2.5: Performance of the systems in CoNLL-2004 task: Semantic role labelling.

system	P	R	F
Hacioglu	72.43	66.77	69.49
Punyakanok	70.07	63.07	66.39
Carreras	71.81	61.11	66.03
Lim	68.42	61.47	64.76
Park	65.63	62.43	63.99
Higgins	64.17	57.52	60.66
Van den Bosch	67.12	54.46	60.13
Kouchnir	56.86	49.95	53.18
Baldewein	65.73	42.60	51.70
Williams	58.08	34.75	43.48
baseline	54.60	31.39	39.87

In CoNLL-2005, the main focus of interest was to increase the amount of syntactic and semantic input information, aiming to boost the performance of machine learning systems on the SRL task. Following earlier editions of the shared task, the input information contained several levels of annotation apart from the role labeling information: words, POS tags, chunks, clauses, named entities, and parse trees. Compared to the shared task of CoNLL-2004, the novelties introduced in the 2005 edition were:

1. The training corpus was substantially enlarged. This allows to test the scalability of learning-based SRL systems to big data sets and compute learning curves to see how much data is necessary to train.
2. Aiming at evaluating the contribution of full parsing in SRL, the complete syntactic trees given by several alternative parsers was provided as input information for the task.
3. In order to test the robustness of the presented systems, a cross-corpora evaluation was performed using fresh test sets from corpora other than the one used for training.

The following are the systems that participated in CoNLL-2005 Semantic Role Labeling (SRL) task.

- Tsai et al. [140] proposed a method that exploits full parsing information by representing it as features of argument classification models and as constraints in integer linear learning programs. The experimental results showed that full parsing information not only increases the F-score of argument classification models by 0.7%, but also effectively removes all labeling inconsistencies, which increases the F-score by 0.64%. The ensemble of SVM and ME also boosts the F-score by 0.77%. The system achieved an F-score of 76.53% on the development set and 76.38% on the Test WSJ.

- Sutton et al. [137] used a cascade of maximum-entropy classifiers which select the semantic argument label for each constituent of a full parse tree. As in other systems, they used three stages: pruning, identification, and classification. First, in pruning, they use a deterministic preprocessing procedure introduced to prune many constituents which are almost certainly not arguments. Second, in identification, a binary MaxEnt classifier is used to prune remaining constituents which are predicted to be null with high probability. Finally, in classification, a multiclass MaxEnt classifier is used to predict the argument type of the remaining constituents.

- Surdeanu et al. [136] used Ada-Boost learning algorithm for the SRL task. They have used Charniak parser to produce parse of the sentence. The labeling problem is modeled using a rich set of lexical, syntactic, and semantic attributes and learned using one-versus-all Ada-Boost classifiers.

- Sameer Pradhan et al. [113] used SVM classifier for semantic role labelling. Features were extracted by first generating the Collins and Charniak syntax trees from the word by word decomposed trees in the CoNLL data. The chunking system for combining all features was trained using a 4-fold paradigm. In each fold, separate SVM classifiers were trained for the Collins and Charniak parses using 75% of the training data. That is, one system assigned role labels to the nodes in Charniak based trees and a separate system assigned roles to nodes in Collins based trees. The other 25% of the training data was then labelled by each of the systems. Iterating this process 4 times created the training set for the chunker. After the chunker was trained, the Charniak and Collins based semantic labelers were then retrained using all of the training data.

- Park et al. [108] used maximum entropy model for the SRL task. They first identified parse constituents that represent valid semantic arguments of a given predicate, and then assigned appropriate semantic roles to the the identified parse constituents. In order to improve the performance of identification, they tried to incorporate clause boundary restriction and tree distance restriction into pre-processing of the identification phase. They have used 19 features for this task.

- Ozgencil et al. [105] proposed a two-layer architecture to first identify the arguments and then to label them for each predicate for the SRL task. The components are implemented as SVM classifiers using libSVM. They have chosen Radial Basis Function (RBF) kernel for this task. They have used 5 features for identification and seven features for role labelling task.

- Alessandro Moschitti et al. [92] used SVMlight-TK software which encodes the tree kernels in the SVM-light software for SRL task. They analyzed the impact of a hierarchical categorization on the semantic role labeling task in this work. They divided the predicate argument labeling in two subtasks: (a) the detection of the arguments related to a target, i.e. all the compounding words of such argument, and (b) the classification of the argument type, e.g. A0 or AM. They used the default polynomial kernel (degree=3) for the linear feature representations and the tree kernels for the structural feature processing.

- Tomohiro Mitsumori et al. [89] used support vector machines for the SRL task. They have used 11 features: words (1st), POS tags (2nd), base phrase chunks (3rd), named entities (4th), token depth (5th), predicate (6th), position of tokens (7th), phrase distance (8th), flat paths (9th), semantic classes (10th), argument classes (11th).

- Aria Haghighi et al. [53] used a joint model that captures dependencies among arguments of a predicate using log-linear models in a discriminative re-ranking framework. They have defined an exhaustive list of 27 features for this task.

- Akshar Bharati et al. [12] used maximum entropy based classifier for SRL task. This approach has two stages: first, identification of whether the argument is mandatory or optional and second, the classification or labelling of the arguments. In the first stage, the arguments of a verb are put into three classes, (1) mandatory, (2) optional or (3) null. The maximum entropy based classifier is used to classify the arguments into one of the above three labels.

- Ting Liuet al. [82] used a maximum entropy classifier for SRL task.
- Peter Koomen et al. [72] used a learning algorithm which is a variation of the Winnow update rule incorporated in SNoW (Sparse Network of Winnows), a multi-class classifier that is tailored for large scale learning tasks.
- Marquez et al. [84] used ensembles of decision trees learned through the Ada-Boost learning algorithm.
- Ponzetto and Strube [112] used decision trees (C4.5) for this task.
- Lin and Smith [79] presented a proposal radically different from the rest, with very light learning components. Their approach (Consensus in Pattern Matching, CPM) contains some elements of Memory-based Learning and ensemble classification.
- Cohn and Blunsom [28] used tree conditional random fields (T-CRF) that extend the sequential CRF model to tree structures.
- Johansson and Nugues [60] used relevant vector machine (RVM), which is a kernel based linear discriminant inside the framework of Sparse Bayesian Learning for the SRL task.
- Szu-ting Yi et al. [151] proposed a system for SRL task by combining two different parsers trained on syntactic constituent information and semantic argument information. Their SRL system has 5 phases: Parsing, Pruning, Argument Identification, Argument Classification, and Post Processing.
- Tjong Kim Sang et al. [139] proposed a system with the combination of different models like maximum entropy models, support vector machines and memory-based learning approach. A novel automatic post-processing procedure based on Levenshtein-distance based correction was able to achieve a performance increase by correcting unlikely role assignments.

Table 2.6: Performance of the systems in CoNLL-2005 task: Semantic role labelling.

System	WSJ			Brown			WSJ+Brown		
	P(%)	R(%)	F	P(%)	R(%)	F	P(%)	R(%)	F
Punyakanok	82.28	76.78	79.44	73.38	62.93	67.75	81.18	74.92	77.92
Haghighi	79.54	77.39	78.45	70.24	65.37	67.71	78.34	75.78	77.04
Marquez	79.55	76.45	77.97	70.79	64.35	67.42	78.44	74.83	76.59
Pradhan	81.97	73.27	77.37	73.73	61.51	67.07	80.93	71.69	76.03
Surdeanu	80.32	72.95	76.46	72.41	59.67	65.42	79.35	71.17	75.04
Tsai	82.77	70.90	76.38	73.21	59.49	65.64	81.55	69.37	74.97
Che	80.48	72.79	76.44	71.13	59.99	65.09	79.30	71.08	74.97
Moschitti	76.55	75.24	75.89	65.92	61.83	63.81	75.19	73.45	74.31
Tjongkimsang	79.03	72.03	75.37	70.45	60.13	64.88	77.94	70.44	74.00
Szu-Ting Yi	77.51	72.97	75.17	67.88	59.03	63.14	76.31	71.10	73.61
Ozgencil	74.66	74.21	74.44	65.52	62.93	64.20	73.48	72.70	73.09
Johansson	75.46	73.18	74.30	65.17	60.59	62.79	74.13	71.50	72.79
Cohn	75.81	70.58	73.10	67.63	60.08	63.63	74.76	69.17	71.86
Park	74.69	70.78	72.68	64.58	60.31	62.38	73.35	69.37	71.31
Mitsumori	74.15	68.25	71.08	63.24	54.20	58.37	72.77	66.37	69.43
Venkatapathy	73.76	65.52	69.40	65.25	55.72	60.11	72.66	64.21	68.17
Ponzetto	75.05	64.81	69.56	66.69	52.14	58.52	74.02	63.12	68.13
Lin	71.49	64.67	67.91	65.75	52.82	58.58	70.80	63.09	66.72
Sutton	68.57	64.99	66.73	62.91	54.85	58.60	67.86	63.63	65.68
Baseline	51.13	29.16	37.14	62.66	33.07	43.30	52.58	29.69	37.95

2.9 Finite State Automata

An automaton is a general term for any formal model of computation. Typically, an automaton is represented as a state machine. Finite State Automaton (FSA) is a model of computation consisting of a finite set of states, a start state, an input alphabet, and a transition function that maps input symbol and current state to next state. A state transition usually has some rules associated with it that govern when the transition may occur.

There are two kinds of Automata called deterministic finite state automata (DFA) and non-deterministic finite state automata (NDFA or NFA).

In deterministic automata, for each state there is exactly one transition for a given input. A deterministic FSA consists of five components $< Q, \sum, q_0, F, \delta >$, where

- Q is a finite set of states
- Σ is a finite nonempty set of symbols: the input alphabet

- q_0 is the start state, $q_0 \in Q$
- F is the set of final states, F $\subseteq$ Q
- δ is the transition function $Q \times \Sigma \rightarrow Q$, where δ(q,a) returns the next state of the automaton when it is in state q and sees the symbol a.

In non-deterministic automaton, there can be none, one or more than one transition from a given state for a given possible input. A nondeterministic FSA consists of five components $< Q, \sum, q_0, F, \delta >$, where

- Q is a finite set of states
- Σ is a finite nonempty set of symbols: the input alphabet
- q_0 is the start state, $q_0 \in Q$
- F is the set of final states, F $\subseteq$ Q
- δ is the transition function $Q \times (\Sigma \bigcup \{\varepsilon\}) \rightarrow 2^Q$, where δ(q,a) returns the set of possible next states when it is in state q and sees the symbol a and ε is the empty string,

In fact, we can always find a deterministic automaton that recognizes/generates exactly the same language as a non-deterministic automaton. There exists an algorithm which can transform any NDFA into an equivalent DFA. This can be performed using the powerset construction which may lead to an exponential rise in the number of necessary states in a DFA.

A finite state automaton is a simple computing machine that has a single tape. An automaton can be said to recognize a string if we view the contents of its tape as input. Alternatively, we can say that an automaton generates strings, which means viewing its tape as an output tape. On this view, the automaton generates a formal language, which is a set of strings. The two views of automata are equivalent: the function that the automaton computes is precisely the indicator function of the set of strings it recognizes. The class of languages generated by finite automata is known as the class of regular languages.

DFAs are one of the most practical models of computation, since there exist a linear time, constant-space algorithm to simulate a DFA for a given stream of input. Given two DFAs there are efficient algorithms to find a DFA recognizing the union, intersection, and complements of the languages they recognize. There are also efficient algorithms to determine whether a DFA accepts any strings, whether a DFA accepts all strings, whether two DFAs recognize the same language, and to find the DFA with a minimum number of states for a particular regular language.

On the other hand, DFAs are strictly limited in power in the languages they can recognize many simple languages, including any problem that requires more than constant space to solve, cannot be recognized by a DFA. The classical example of a simply described language that no DFA can recognize is the language consisting of strings of the form a^nb^n some finite number of a's, followed by an equal number of b's. It can be shown that no DFA can have enough states to recognize such a language.

A Finite State Transducer (FST) is a finite state machine with two tapes in contrast with an ordinary finite state automaton, which has a single tape. The two tapes of a transducer are typically viewed as an input tape and an output tape. On this view, a transducer is said to transduce (i.e., translate) the contents of its input tape to its output tape, by accepting a string on its input tape and generating another string on its output tape. It may do so nondeterministically and it may produce more than one output for each input string. A transducer may also produce no output for a given input string, in which case it is said to reject the input. In general, a transducer computes a relation between two formal languages. The class of relations computed by finite state transducers is known as the class of rational relations.

An FST is a 7-tuple $< Q, \Sigma_1, \Sigma_2, \delta, q_0, F, \sigma >$ such that:

- Q is a finite set of states
- Σ_1 is a finite set, the inputalphabet
- Σ_2 is a finite set, the outputalphabet
- δ is the transition function $Q \times (\Sigma \bigcup \{\varepsilon\}) \rightarrow 2^Q$
- $q_0 \in$ Q is the start state
- F is the set of final states, F $\subseteq$ Q

- σ is the outputfunction $\sigma : Q \times (\Sigma_1 \bigcup \{\varepsilon\}) \times Q \rightarrow \Sigma_2^*$

2.9.1 Brief Survey on Finite State Machine based Parsing Systems

Steven Abney [3] used in 1996 cascades of finite state machines for parsing unrestricted text. Cascading of finite state machines means combining them by adding arcs between the FSAs such that transitions can be made from one FSA into the next one. He showed that deterministic parsers specified by finite state cascades are fast and reliable. They can be extended at modest cost to construct parse trees with finite feature structures. Finally, such deterministic parsers do not necessarily involve trading off accuracy against speed they may in fact be more accurate than exhaustive search stochastic context free parsers. Only preliminary evaluation has been done on manually parsed samples. G. Grefenstette [49], in 1996, used finite state transducers for identifying noun groups and verb groups.

Fabio Ciravegna and Alberto Lavelli [26], in 2002, proposed a robust approach to parsing suitable for Information Extraction (IE) from texts using finite-state cascades. The approach is characterized by the construction of an approximation of the full parse tree that captures all the information relevant for IE purposes, leaving the other relations under-specified. Sequences of cascades of finite-state rules deterministically analyze the text, building unambiguous structures. Initially basic chunks are analyzed; then clauses are recognized and nested; finally modifier attachment is performed and the global parse tree is built. The approach has been tested for Italian, English, and Russian. A parser based on such an approach has been implemented as part of Pinocchio, an environment for developing and running IE applications.

Gondy Leroy et al. [47], in 2003, used cascaded finite state automata for identifying entities and relations in biomedical text. These entities and relations are usually pre-specified entities, e.g., proteins, and pre-specified relations, e.g., inhibit relations. Cascaded finite state automata identifies the relations between individual entities. The automata are based on closed-class English words and model generic relations not limited to specific words. The parser also recognizes coordinating conjunctions and captures negation in text, a feature usually ignored by others. Three cancer researchers evaluated 330 relations extracted from 26 abstracts of interest to them. There were 296 relations correctly extracted from the abstracts resulting in 90% precision of the relations and an average of 11 correct relations per abstract.

Cascades of finite state transducers are extensively used in many other language processing applications such as for obtaining verb sub-categorization [6], information extraction [40] etc.

2.10 Hidden Markov Models

Hidden Markov Models (HMMs) are powerful statistical models for modeling sequential or time-series data, and have been successfully used in many tasks such as speech recognition, part of speech tagging, shallow parsing, protein/DNA sequence analysis, robot control etc.

A Hidden Markov Model[96] is a finite set of states, each of which is associated with a probability distribution. Transitions among the states are governed by a set of probabilities called transition probabilities. In a particular state an outcome or observation can be generated, according to the associated probability distribution. In a regular Markov model, the states are directly visible to the observer, and therefore the state transition probabilities are the only parameters. In a Hidden Markov Model, it is only the outcome, not the state, which is visible to an external observer. The states are "hidden". Hence the name Hidden Markov Model.

A HMM is characterized by the following [117]:

1) *N*, the number of states in the model.

2) *M*, the number of distinct observation symbols associated with a state, i.e., the discrete alphabet set.

3) The state transition probability distribution $A = a_{ij}$, where,

$$a_{ij} = P(q_{t+1} = j | q_t = i), \qquad 1 \leq i, j \leq N \tag{2.5}$$

with the state transition probabilities satisfying the constraints

$$\begin{aligned} a_{ij} &\geq 0 \\ \sum_{j=1}^{N} a_{ij} &= 1 \end{aligned}$$

4) The observation symbol probability distribution in state j, $B = \{b_j(k)\}$, where,

$$b_j(k) = P\left[v_k | q_t = j\right], \quad 1 \leq j \leq N \qquad 1 \leq k \leq M \tag{2.6}$$

and satisfying the following constraints

$$\begin{aligned} b_j(k) &\geq 0 \\ \sum_{k=1}^{M} b_j(k) &= 1 \end{aligned}$$

5) The initial state distribution $\pi = \pi_i$, where

$$\pi_i = P\left[q_1 = i\right], \quad 1 \leq i \leq N \tag{2.7}$$

satisfying the constraint $\sum_{i=1}^{N} \pi_i = 1$

A complete specification of an HMM requires specification of two model parameters (N and M), specification of observation symbols, and the specification of the three probability measures A,B, and π. For convenience, we use the compact notation for the entire model λ can be denoted as :

$$\lambda = (A, B, \pi) \tag{2.8}$$

Three Basic Assumptions:

- First order Markov assumption: The probability of a certain observation at time t depends only the previous observation at time t-1 rather than the whole history.

$$P(q_t = j | q_{t-1} = i, q_{t-2} = h, ...q_1 = a) = P(q_t = j | q_{t-1} = i) \tag{2.9}$$

Generally the first order markov assumption is considered in building HMMs and the resulting model becomes actually a first order HMM.

- The stationarity assumption: Each probability in the state transition matrix is time independent - that is, the matrices do not change with time. In practice, this can be one of the most unrealistic assumptions of Markov models about real processes.

$$P(q_{t_1+1} = j | q_{t_1} = i) = P(q_{t_2+1} = j | q_{t_2} = i) \tag{2.10}$$

- The output independence assumption: This is the assumption that current output (observation) is statistically independent of the previous outputs (observations). We can formulate this assumption mathematically, by considering a sequence of observations, $O = (O_1, O_2, \ldots, O_T)$

  Then according to this assumption for an HMM

$$P(O|q_1, q_2, ...q_T, \lambda) = \Pi_{t=1}^{T} P(O_t|q_t, \lambda) \tag{2.11}$$

  However unlike the other two, this assumption has a very limited validity. In some cases this assumption may not be fair enough and therefore becomes a severe weakness of the HMMs.

Three Basic Problems:

1. Given the observation sequence $O = (O_1, O_2, \ldots, O_T)$ and a model $\lambda = (A, B, \pi)$, how do we efficiently compute $P(O|\lambda)$, the probability of the observation sequence, given the model?
2. Given the observation sequence $O = (O_1, O_2, \ldots, O_T)$ and the model λ, how do we choose a corresponding state sequence $Q = (q_1, q_2, \ldots, q_t, \ldots, q_T)$ which is optimal in some meaningful sense ("that best explains the observations")?
3. Given the observation sequence $O = (o_1, o_2, \ldots, o_T)$, how do we adjust the model parameters $\lambda = (A, B, \pi)$ to maximize $P(O|\lambda)$?

Problem 1 is the evaluation problem, namely given a model and a sequence of observations, how do we compute the probability that the observed sequence was produced by the model. We can also view the problem as one of scoring how well a given model matches a given observation sequence. For example, if we consider the case in which we are trying to choose among several competing models, the solution to problem 1 allows us to choose the model which best matches the observations. Enumerating all possible state sequences and then computing the probability of observing the given sequence using that state sequence is computationally too expensive. There are two algorithms called the *Forward Algorithm* and the *Backward Algorithm* which can do the same computation more efficiently.

Problem 2 is the one in which we attempt to uncover the hidden part of the model, i.e., to find the "best" state sequence. We need to use some optimality criterion. There are several reasonable optimality criteria that can be imposed, and hence the choice of criterion is dependent on the intended use for the uncovered state sequence. The solution to this problem two is given by *Viterbi Algorithm.* In the case of POS tagging, we could model states as POS tags and words as observation symbols. POS tagging would then correspond to identifying an optimal state sequence using this Viterbi algorithm.

In problem 3 we attempt to optimize the model parameters so as to best describe a given observation sequence. The observation sequence used to adjust the model parameters is called a training sequence. The training problem is a crucial one for most applications of HMMs, since it allows us to optimally adapt model parameters to observed training data. The *Baum-Welch Algorithm* provides a method for doing this. This is basically a parameter re-estimation technique. Instead of simply computing empirical probabilities from training corpora, we can start with an initial model and iteratively refine the parameters as we get to see more and more training data. The Baum-Welch algorithm is actually an application of the EM algorithm.

2.10.0.1 Problem 1

Let the state sequence q is

$$Q = (q_1, q_2 \cdots q_t) \tag{2.12}$$

For a fixed state sequence, a most straight forward way to determine the

$P(O,Q|\lambda)$ is

$$P(O,Q|\lambda) = P(O|Q,\lambda)P(Q|\lambda) \tag{2.13}$$

where

$$P(O|Q,\lambda) = b_{q_1}(O_1)b_{q_2}(O_2)\cdots b_{q_t}(O_t) \tag{2.14}$$

$$P(Q|\lambda) = \pi_{q_1}a_{q_1,q_2}a_{q_2,q_3}\cdots a_{q_{t-1},q_t} \tag{2.15}$$

Finally, the $P(O,Q|\lambda)$ can be written as

$$P(O,Q|\lambda) = \pi_{q_1}b_{q_1}(O_1)a_{q_1,q_2}b_{q_2}(O_2)a_{q_2,q_3}\cdots a_{q_{t-1},q_t}b_{q_t}(O_t) \tag{2.16}$$

We wish to calculate probability of observation sequence, O $= O_1, O_2, ..., O_t$, given the model λ, i.e. $P(O|\lambda)$. The straight forward way of calculating this is through enumerating every possible state sequence of length T.

The probability of O(given the model) is obtained by summing this joint probability over all possible state sequences.

$$P(O|\lambda) = \sum_{allQ} P(O|q,\lambda)P(q|\lambda) = \sum_{q_1,q_2,\dots q_t} \pi_{q_1}b_{q_1}(O_1)a_{q_1,q_2}b_{q_2}(O_2)a_{q_2,q_3}\cdots a_{q_{t-1},q_t}b_{q_t}(O_t) \tag{2.17}$$

Since at t=1,2,...,T, there are N possible states that can be reached, the number of possible state sequences will become N^T. For each state sequence, there are 2T multiplications required for each term in the above equation. Hence the direct computation of the above equation involves 2TN^T multiplications, which is computationally not feasible, even for small values of N and T.

1. **Forward Algorithm**

   A better approach is to recognize that many redundant calculations are involved in direct calculation of $P(O,Q|\lambda)$ and therefore caching calculations can lead to reduced complexity. This can be done in the following way:

   Consider the forward variable $\alpha_t(i)$ defined as

   (a) Initialization:

   $$\alpha_t(i) = \pi_i b_j(O_1), 1 \leq i \leq n \tag{2.18}$$

(b) Induction:

$$\alpha_{t+1}(j) = [\sum_{i=1}^{n} \alpha_t(i)a_{ij}]b_j(O_{t+1}), 1 \leq i \leq n, 1 \leq t \leq T-1, \tag{2.19}$$

(c) Termination:

$$P(O|\lambda) = \sum_{i=1}^{n} \alpha_T(i) \tag{2.20}$$

The induction step is the key to the forward algorithm. In forward algorithm, we can exploit knowledge of the previous step to compute information about a new one i.e. for each state j , $\alpha_j(t)$ stores the probability of arriving in that state having observed the observation sequence up until time t. It is apparent that by caching α values the forward algorithm reduces the complexity of calculations involved to N^2T rather than $2TN^T$.

2. **Backward Algorithm**

Since HMMs satisfy the Markov property, we can also calculate the probability of observing O given a particular model by working backward from the end of the sequence. In similar way to the forward algorithm, the backward variable $\beta_t(i)$ can be defined as

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, ..., O_T|q_t = i, \lambda) \tag{2.21}$$

i.e. the probability of partial observation sequence from t+1 to end, given state i and model λ. We can solve for $\beta_t(i)$ inductively as follows:

(a) Initialization:

$$\beta_T(i) = 1, 1 \leq i \leq n \tag{2.22}$$

(b) Induction:

$$\beta_t(i) = \sum_{j=1}^{n} a_{ij}b_j(O_{t+1})\beta_{t+1}(j), 1 \leq i \leq n, t = T-1, T-2, ...1 \tag{2.23}$$

(c) Termination:

$$P(O|\lambda) = \sum_{i=1}^{n} \pi_i b_i(O_1)\beta_1(i) \tag{2.24}$$

2.10.0.2 Problem 2

Here, we wish to find the "optimal" state sequence associated with the given observation sequence. The difficulty here lies with the definition of optimal state sequence i.e. there may be several possible optimality criteria. For example, one possible optimality criterion is to choose the states q_t which are individually most likely. This optimality criterion maximizes the expected number of correct individual states. But this approach has some problem with the resulting state sequence. When the HMM state transitions has zero probability, the optimal state sequence may not be a valid sequence at all. This is due to the fact that the algorithm simply determines the most likely state at every instant, without regard to the probability of occurrence of sequence of states.

1. **Viterbi Algorithm**

   To find the single best state sequence, Q $= q_1, q_2 \cdots q_T$, for the given observation sequence O $= o_1, o_2 \cdots o_t$, we need to define the quantity,

$$\delta_t(i) = max_{q_1,q_2 \cdots q_{t-1}} P[q_1, q_2 \cdots q_t = i, O_1, O_2 \cdots O_t|\lambda] \qquad (2.25)$$

   i.e., $\delta_t(i)$ is the best score along a single path, at time t, which accounts for the first t observations and ends in state S_i. By induction, we have

$$\delta_{t+1}(j) = [max_i \delta_t(i) a_{ij}].b_j(O_{t+1}) \qquad (2.26)$$

   To actually retrieve the state sequence, we need to keep track of the argument which maximizes the above equation, for each t and j. We do this via the array $\psi_t(j)$. The complete procedure for finding the best state sequence is given below:

   1) **Initialization Step:**

$$\delta_t(i) = \pi_i b_i(O_1, 1 \leq i \leq n \qquad (2.27)$$

$$\psi_1(i) = 0 \qquad (2.28)$$

2) **Recursion Step:**

$$\delta_t(j) = max_{1 \leq i \leq n}[\delta_{t-1}(i)a_{ij}]b_j(O_t), 2 \leq i \leq T, 1 \leq j \leq n \tag{2.29}$$

$$\psi_t(j) = argmax_{1 \leq i \leq n}[\delta_{t-1}(i)a_{ij}] \tag{2.30}$$

3) **Termination Step:**

$$P^* = max_{1 \leq i \leq n}[\delta_T(i)] \tag{2.31}$$

$$q_T^* = argmax_{1 \leq i \leq n}[\delta_T(i)] \tag{2.32}$$

$$q_t^* = \psi_{t+1}(q_{t+1}^*), t = T-1, T-2, ...1 \tag{2.33}$$

It can be noted that the Viterbi algorithm is similar in implementation to the forward algorithm. Instead of summing up all the paths, the computation is done by picking the best path in each cell. Hence, the complexity for Viterbi algorithm is also $O(N^2T)$.

2.10.0.3 Problem 3

Here, we need to find a method for adjusting the model parameters (A,B,π) to maximize the probability of the given observation sequence given the model. There is no known way to analytically solve for the model which maximizes the probability of the observation sequence. In fact, given any finite observation sequence as the training data, there is no optimal way of estimating the model parameters. But we can choose λ=(A,B,π) such that $P(O|\lambda)$ is locally maximized using an iterative procedure called Baum-Welch method (a variant of Expectation-Maximization algorithm) or using gradient descent techniques. Here we describe Baum-Welch method for re-estimation of parameters.

1. **Baum-Welch Algorithm**

   Baum-Welch algorithm [41] is an iterative process for estimating HMM parameters. The Baum-Welch algorithm starts from an initial model and iteratively improves on it until convergence is reached. This algorithm maximizes $P(O|\lambda)$ by adjusting the parameters of λ. This optimization criterion is called maximum likelihood criterion and $P(O|\lambda)$ is called the likelihood function. The algorithm is guaranteed to converge to an HMM that locally maximizes the likelihood (the probability of the training data given the model). Since the Baum-Welch algorithm is a local iterative method, the resulting HMM and the number of required iterations depend heavily on the initial model.

In order to describe the procedure for re-estimation of HMM parameters, we first define $\xi_t(i,j)$, the probability of being in state i at time t and state j at time t+1, given the model and observation sequence.

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j|O,\lambda) \tag{2.34}$$

Using Bayes law, it can be written as

$$\xi_t(i,j) = \frac{P(q_t = i, q_{t+1} = j, O|\lambda)}{P(O|\lambda)} \tag{2.35}$$

The numerator can be written as

$$P(q_t = i, O_1, O_2...O_t, O_{t+1}, ...O_T, q_{t+1} = j|\lambda) \tag{2.36}$$

$$= P(q_t = i, O_1, O_2...O_t|\lambda)P(O_{t+1}, O_{t+2}, ...O_T, q_{t+1} = j|\lambda)(MarkovianProperty) \tag{2.37}$$

$$P(O_{t+1}, O_{t+2}, ...O_T, q_{t+1} = j|\lambda) = P(q_{t+1} = j, O_{t+1}|\lambda)P(O_{t+1}, O_{t+2}, ...O_T|q_{t+1} = j, \lambda) \tag{2.38}$$

$$= a_{ij}b_j(O_{t+1})\beta_{t+1}(j) \tag{2.39}$$

By using the definitions of forward and backward variables, we can write $\xi_t(i,j)$ in the form

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{P(O|\lambda)} = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)} \tag{2.40}$$

If we sum up $\gamma_{t=1}(i)$ from t=1 to T we get a quantity which can be viewed as the expected number of times state i is visited or if we sum up only up to T-1 then we shall get the expected number of transitions out of state i. Similarly, if $\xi_t(i,j)$ be summed up from t=1 to T-1, we shall get the expected number of tanstions from the state i to state j. Hence

$\sum_{t=1}^{T-1}\gamma_t(i)$ = expected number of times state i is visited

where $\gamma_{t=1}(i)$=$\sum_{i=1}^{n}\xi_t(i,j)$

$\sum_{t=1}^{T-1}\xi_t(i,j)$= expected number of transitions from state i to state j

Now we can write Baum-Welch re-estimation formulas as:

$\overline{\pi_i}$ = expected frequency in state i at time (t = 1) = $\gamma_t(i)$

$\overline{a_{ij}}$= expected number of transition from state i to state j/(expected nubmer of transitions from state i

$$= \frac{\sum_{t=1}^{T-1}\xi_t(i,j)}{\sum_{t=1}^{T-1}\gamma_t(i)} \tag{2.41}$$

$\overline{b_j(k)}$= expected number of times in state j and observing symbol k/expected number of times in state j

$$= \frac{\sum_{t=1,O_t=k}^{T}\gamma_t(j)}{\sum_{t=1}^{T}\gamma_t(j)} \tag{2.42}$$

A serious problem with any hill-climbing optimization technique is that it often ends up in a local maximum. The same is true for the forward-backward procedure used to estimate HMMs by maximizing the likelihood (or the a posteriori model probability). In fact, it will almost always end up in a local maximum.

If we give more and more training data, we will get an HMM that is pretty close to optimal for the given training sequences. It must be noted, however, that any gradient descent algorithm is not guaranteed that the HMM will end up in a state of globally minimal error. Instead, it settles into a local minimum, which hopefully is not too far from the global minimum. To deal with this problem, we train the algorithm several times from different initial models. The resulting models then represent different local maxima, and we pick the one with the highest likelihood.

One final point that should be remembered is that the computation of the forward and backward probabilities involves taking the product of a large number of probabilities. In practice, this means that the actual numbers involved become very small. Hence, to avoid numerical problems, the forward-backward computation can be done better in log arithmetic.

2.10.1 Brief Survey on Hidden Markov Models based works in Parsing

Molina and Pla[91], in 2002, presented a shallow parser based on Hidden Markov Models (HMMs). HMMs were used to find the most probable sequence of output shallow parsing labels for the current sequence of inputs. In their model, they used information about the whole sentence into account when determining the output shallow parsing label for each word, since it is the probability of the whole sequence of output tags occurring given the current input that is maximized (and not just the probability of individual decisions). The authors used second order HMMs to a variety of shallow parsing tasks.

Wide R. Hogenhout and Yuji Matsumoto, in 1998, proposed a model for statistical parsing using Hidden Markov Models (HMMs).

2.11 Best First Search Strategies

Best first search is a way of combining the advantages of both depth-first and breadth-first search into a single method. At each step of the best-first search process, we select the most promising of the nodes we have generated so far. This is done by applying an appropriate heuristic function to each node. We then expand the chosen node by using rules to generate its successors. If one of them is a solution, we can quit. If not, all these new nodes are added to the set of nodes generated so far. Again the most promising node is selected and the process continues. If a solution is not found, that branch may start to look less promising than one of the top-level branches that has so far been ignored. At that point, the new more promising but previously ignored branch will be explored. But the old branch is not forgotten. It remains in the set of generated but unexplored nodes. The search can return to it whenever all the others get bad enough that it is again the most promising node.

If one has a good evaluation function, best first search may drastically cut down the amount of search to find a solution. One may not find the best solution, but if a solution exists you will eventually find it, and there is a good chance of finding it quickly. Of course, if your evaluation function is no good then it is as good as using simpler search techniques such as depth first or breadth first. And if evaluation function is very expensive or complex the benefits of cutting down on the amount of search may be outweighed by the costs of assigning a score.

Beam search is a heuristic search algorithm that is an optimization over best-first search. Like best-first search, it uses a heuristic function to estimate the promise of each node it examines. Beam search, however, only unfolds the first m most promising nodes at each depth, where m is a fixed number, the "beam width." While beam search is space-bounded as a function of m, it is neither optimal nor complete while m is finite. As m increases, beam search approaches the functionality of best-first search. Initially m random states are chosen. The successors of these m states are all calculated. If the Goal Node is reached, the algorithm halts. Else the best m states of these successors are taken and the steps repeated.

A* is a graph search algorithm that finds a path from a given initial node to a given goal node (or one passing a given goal test). It employs a "heuristic estimate" that ranks each node by an estimate of the best route that goes through that node. It visits the nodes in order of this heuristic estimate. The A* algorithm is therefore an example of best-first search.

The A* Best First Search Strategy combines two factors, namely, effort already spent in pursuing the current path (g), and, estimated effort required to reach the goal state (h). A single combined measure of goodness (f) is computed for each node in the search tree and the best node is selected for subsequent expansion: $f(n) = g(n) + h(n)$. In our case, the effort already spent can obtained from the probabilities given by the HMM module. The distance to the goal node can be estimated in terms of the words yet to be covered in the given sentence.

2.11.1 Brief Survey of Search Algorithms used for Selecting Best Parse

Natural languages abound in lexical and structural ambiguities and parsing is exponential in complexity. Several types of methods for accelerating parse selection have been proposed. Roark [121] and Ratnaparkhi [119] used a beam-search strategy, in which only the best n parses are tracked at any moment. Parsing time is linear and can be made arbitrarily fast by reducing n. This is a greedy strategy, and the actual Viterbi (highest probability) parse can be pruned from the beam because, while it is globally optimal, it may not be locally optimal at every parse stage. Charniak et al. [21] described best-first chart parsing which attempts to parse efficiently by working on the edges that are judged 'best' by some probabilistic figure of merit. This approach dramatically reduces the work done during parsing, though it gives no guarantee that the first parse returned is the actual Viterbi parse. In their work, they found that exhaustively parsing maximum 40-word sentences from the Penn Treebank II requires an average of about 1.2 million edges per sentence [21]. Dan Klein and Christopher D. Manning [70] presented an extension of the classic A* search procedure to tabular PCFG parsing. The use of A* search can dramatically reduce the time required to find a best parse by conservatively estimating the probabilities of parse completions. On average-length Penn Treebank sentences, the most detailed estimate reduces the total number of edges processed to less than 3% of that required by exhaustive parsing. Unlike best-first and finite-beam methods for achieving this kind of speed-up, an A* method is guaranteed to find the most likely parse, not just an approximation. This parser, which is simpler to implement than an upward-propagating best-first parser, is correct for a wide range of parser control strategies and maintains worst-case cubic time.

Chapter 3

Universal Clause Structure Grammar

Universal Clause Structure Grammar (UCSG) is a framework for parsing natural language sentences that was initiated in the early 90's at University of Hyderabad by Kavi Narayana Murthy[93]. The primary goal of UCSG was to develop a computational architecture that leads to simple grammars and efficient parsing for both positional and relatively free word order languages. In this chapter we describe overall architecture of the UCSG syntax.

3.1 Why UCSG?

From the survey on grammar formalisms [93], it can be seen that many of the grammar formalisms work with phrase structure rules or equivalent and produce parse trees as output. These trees are ordered trees - the order of child nodes is important. Linear order is an innate property of Phrase Structure rules (Context Free or otherwise). Word order is an inalienable aspect of all grammar formalisms that use (ordered) trees or phrase structure rules at any level. A variety of extensions and modifications have been proposed for dealing with languages where order of words is not so significant. To a large extent, these techniques, designed as an after thought, are roundabout and unnecessarily complicated.

On the other hand, traditional grammars for Indian Languages including that of Panini for Sanskrit, dwell much more on agreement and compatibility at both superficial and deeper, semantic sense and much less on word order. These formalisms cannot be directly applied to positional languages without risking the same kind of after-thought adjustments and extensions that are not entirely motivated by theoretical considerations. Strict word order is such a fundamental property of positional languages that it seems almost impossible to separate this aspect out. It appears that none of the grammar formalisms proposed so far are equally well suited for free word order languages exemplified by Indian languages on the one hand, and, positional languages such as English on the other hand. What exactly is common between all these languages which appear to be so very different on the surface? UCSG was motivated by these considerations. Recently, some of the other grammar formalisms like Dependency Grammars and Categorial Grammars have also claimed that they are equally suited for positional and Free word Order Languages.

UCSG aims at understanding what is really the underlying, universal, core of all languages and separate this core from the more superficial differences between various language or language families. The so called relatively free word order languages have absolutely no greater freedom of word order than strict word order languages as far as chunks (basic, non-recursive, non-overlapping phrases) are concerned. It is in fact only entire chunks (also termed word-groups) that can actually move around within a sentence. Further, phrases respect clause boundaries as a matter of principle and do not generally trespass clause boundaries. The real underlying universal core has not so much to do with chunks as it has with clauses. The structure of clauses within a sentence is an extremely important aspect that has somehow not received the kind of attention and focus it deserves in grammar formalisms. Universal Clause Structure Grammar (UCSG) claims that clause structure is a central and universal aspect of sentence structure. Hence the name.

We describe main ideas behind UCSG in the following sections.

3.1.1 What Constitutes a Good Grammar Formalism?

According to UCSG [96], a good grammar formalism should satisfy the following requirements.

3.1.1.1 Simple Grammars

NLP requires exhaustive and precise grammars. Computers have no commonsense and no rule can be taken for granted, however simple or obvious it may appear to be for us. Writing such exhaustive and precise grammars is not a simple task. Complexity grows exponentially with the size of the grammar. A good grammar formalism requires only a small number of simple grammar rules thus simplifying the grammar writing task. A smaller and simpler grammar also makes the parser more efficient because at each step there are only a few alternatives to consider. An efficient parser is required not only because of practical needs for fast processing but also from theoretical point of view. Many of the grammar formalisms proposed by linguists are computationally much more complex than needed. Positing an unnecessarily complex mechanism is unintelligent, unwise and wasteful.

3.1.1.2 Universality

It has been well recognized that despite superficial differences, human languages share certain common underlying principles. A good grammar formalism should lay primary emphasis on the universal aspects of grammar and relegate the treatment of idiosyncratic features of specific languages to a secondary level. The idea is to concentrate more on the rules rather than the exceptions. It is not as important to deal with the idiosyncratic, peculiar and infrequent constructions in specific languages as it is to deal with the more basic, common and frequent constructions elegantly and efficiently. Knowing the rule from the exception is extremely important for getting insights into the true nature of human languages and for discovering the universal nature of our languages.

3.1.1.3 Good Structural Descriptions

One of the most commonly used representations of syntactic structure is the tree. Many grammar formalisms use tree structures to depict the syntactic structure of sentences. A tree structure produced by a syntactic analyzer, also called a parser, is called a parse tree. The children of a node in a tree are also trees by definition. Thus trees show the nested, part-whole, or hierarchical relationships between different constituents. Trees used in NLP and linguistics are ordered trees - the linear order of the nodes in the tree is significant. Trees thus show linear as well as hierarchical structure of sentences.

In many languages, the linear ordering of nodes is sometimes significant and sometimes insignificant. These languages are called relatively free word order languages. Indian languages belong to this type. In a Telugu sentence, for example, all possible permutations of nouns are syntactically valid but typically the verb comes at the right end. Also while the nouns in a clause can come in any order, all of them must necessarily come before the nouns in the following clause. Trees are either ordered trees or unordered trees. Interpreting trees as ordered in some places and unordered elsewhere is not easy. Trees originate from phrase structure rules which imply strict order or constituents. Phrase structure rules and the tree structures they produce are not always the best choices.

Apart from depicting the linear and hierarchical structure of sentences, a syntactic analyzer must also determine the roles played by the various constituents in the sentence. The functional structure of a sentence depicts the assignment of functional or thematic roles to the various constituents in the sentence. Role assignment depends on functional structure constraints such as linear position, morphological inflections, agreement, sub-categorization and selectional restrictions. A noun phrase may be the subject of one clause and the object of another embedded clause. Trees cannot depict functional dependencies. So trees are often annotated and augmented with special links connecting various nodes in order to indicate functional dependencies. The resulting structures are really no longer trees but much more complex structures (such as graphs).

Trees have several other demerits. A parse tree is a monolithic structure that includes units at different levels. A tree is a mess of words, phrases, clauses and the entire sentence. Consequently, parts which logically form one group are thrown far apart. All the problems of long distance dependencies originate from this. Also, trees tend to become very large and unwieldy for long and complex sentences. Every elementary subtree, that is, a subtree that includes just one node and its children, corresponds to one application of a phrase structure rule. Thus trees are closer to phrase structure rules than to the structure of the sentences. We need structural descriptions which separate out and vividly show linear, hierarchical as well as functional structure inherent in the given sentence. Trees are not the most suitable structures for depicting the structure of natural language sentences. Nor are phrase structure rules always the best.

3.1.2 Beyond Phrase Structure Grammars

Natural languages are largely context free. In fact, there are aspects of natural language syntax that do not even require context free power, regular grammars are sufficient. That is, CFGs impose *more structure* than really exists in some cases and *less structure* than required in others. CFGs impose linear structure when not appropriate, as in the case of relatively free word order languages. They impose too much of hierarchical structure in cases such as unbounded branching (where simple repetition, not recursion, is required). As far as dependencies between different constituents are concerned, CFGs are unable to capture the required structural constraints effectively.

What then is good about CFGs? CFGs are excellent for handling situations where both linear and hierarchical aspects of structure are involved and nothing else is important. If hierarchical structure is not involved, we may not even require CFGs and if linear structure is itself not significant, CFGs are no good. CFGs are not the best in any situation where any kind of direct dependency between different constituents is involved. It should also be noted that going beyond CFGs and using more powerful phrase structure rules is not necessarily going to solve all these problems.

Given these strengths and limitations of CFGs, both linguists and computer scientists have endeavored to develop better grammar formalisms for handling natural languages. It would be instructive to view each of these grammar formalisms in terms how exactly they have attempted to overcome the deficiencies of CFGs and improve further. Despite the fact that several grammar formalisms have already been developed, the search for better formalisms continues. Theoretically, the available grammar formalisms are not fully satisfactory in explaining the human language faculty in all its ramifications. Computationally, the belief and hope that simpler and more efficient techniques exist, provides motivation for further search for better and better formalisms.

There is a direct relationship between the generative capacity - the kinds of sentence structures which a grammar can generate or parse, and computational complexity. The most general grammar takes the largest amount of computational resources (processing time and/or memory space), and the least general grammar can be parsed with the least computing resources. It is important to note that these differences in computational complexity are not small differences. They are orders of magnitude differences. It has been natural in NLP, therefore, to look for the least powerful grammar that is sufficient for dealing with natural language sentences. Using more powerful grammars than required is wasteful, inefficient, unintelligent and thus unacceptable by both theoretical and practical considerations.

From the history of the theories and models of natural language syntax, it appears that as soon as the limitations of a particular type of grammar are realized, researchers tend to jump to the next higher level of complexity. CFGs were insufficient and hence computer scientists developed ATN grammars. At the same time linguists developed the Transformational Grammars. Both of these have Type 0 power - much more than needed and much more than can be computationally handled efficiently in practice. A much better strategy [93, 96] to use, perhaps is to think of an appropriate way of breaking the problem into subproblems and employing the least powerful grammar that is essential for each.

3.1.3 Process View and the Importance of Efficiency

It is a central working hypothesis of the linguistic theories that a non-processing characterization is desirable. Grammars can then be developed independent of how they are going to be used by parsers and generators. Details of the algorithms can be kept separate from the basic characterization of the knowledge of language. Grammars can be viewed as a specification of a space of grammatical possibilities that does not say anything about how to search that space. While this division of knowledge into *what* and *how* is a boon to the grammar writer, who can write his rules without worrying about the details of the parser, a purely abstract specification oriented view of grammar can be a very bad choice in NLP.

While linguistics has typically dealt with characterization of structures, the science of computation deals with theories of processing. NLP attempts to build computational models of the processes of understanding and synthesis. Computational paradigm is based on the belief that by developing theories of the processes involved, we will have a clear and revealing way of explaining the structure. The structures follow from the processes. A procedural view point is desirable.

One should be careful not to confuse between idealization and proceduralization. One should not mix up the issue of procedural versus nonprocedural approach with the distinction between competence and performance. Chomsky argues for a model of competence and against a performance model by talking about memory limitations, changes in intention during speech, and even physical states such as coughing, sleepiness or drunkenness. Nobody may want to build a model of performance that includes coughing and sneezing. Idealizations are done in all approaches. This is not any significant argument against the procedural view at all.

Ignoring all purely physical and psychological influences of actual performance, the remaining *knowledge of language*, the characterization of the mental competence of a speaker, also has a procedural part in it that has got to be dealt with explicitly. The knowledge of language includes not only rules, constraints and principles but also the procedural aspects of what rules, constraints or principles are applied, how, when, in what order, etc. We have seen that when people speak, they start from their aims and goals, they build up their strategies, decide the structures and words and construct natural language utterances. Similarly understanding language requires many steps. A procedural view point is both natural and essential for NLP.

It should be emphasized that knowledge does not always have to be declarative and nonprocedural. The *procedure* for multiplying two numbers is very much a part of the *knowledge* of most of us. There was a time in the history of AI where the arguments for and against procedural and declarative representations of knowledge had taken almost the shape of a controversy. It is universally agreed now that both kinds of representations of knowledge have their due share. There is no clash between the two, one complements the other.

There is no guarantee that a good purely declarative competence model automatically leads to efficient performance. A model cannot be judged purely based on the structure of the declarative grammar that it employs. A model would be good only if both the declarative and procedural components are good. We all agree that human beings are efficient processors of language and linguistics is concerned with making good models of human language cognition. How then can linguistic theories ignore the efficiency of performance? It is not enough if grammars are simple and elegant, they must also be efficient. Grammars must be *designed* for efficient processing. Efficiency is a term applicable to procedures not abstract declarative knowledge.

Thus our aim in NLP should be to develop simple and elegant grammars that are also amenable for efficient parsing. We cannot characterize grammars independent of how they are going to be used. Parsing techniques and grammars are closely tied up. Data structure and algorithms are two sides of the same coin. We have seen that developing grammars is a challenging task. Thus ease of grammar development is also a key issue.

Modern linguists within the generative tradition believe that there is a single universal grammar underlying all human languages. Children all over the world pick their language with more or less equal ease and within more or less the same amount of time. How do we explain this? There must be some underlying universal principles and all the differences we see across languages of the world must be superficial differences that can be handled by setting values for some parameters. The parameters themselves must also be universal. A great deal of research has gone on within this tradition over the last 50 years or so. However, neither the universal grammar itself or its instantiations into any specific language seem to have been developed into an exhaustive, detailed and precise enough description which can be applied within a computational framework for any given language for NLU or NLG.

UCSG system was developed within the computer science community and developing a wide coverage, robust and computationally efficient system was as much a major goal as developing a theory of syntax per se. This contrasts with systems that were initially developed with purely linguistic aims and objectives and moulded into a computational framework at some later point of time. In fact UCSG never made any strong theoretical claims on the nature of human language processing - how a child acquires language or other such aspects that form the essence of modern generative linguistics. The main concern of UCSG has been how to divide the problem of syntactic analysis appropriately so that we get a computationally viable, universal and efficient parser.

In the next few sections, we describe the important aspects of UCSG full parsing architecture. We describe the different modules of UCSG full parsing architecture. The primary goals of UCSG full parsing architecture are:

1. Computational Viability: Computational Grammars need to be exhaustive, robust, precise and amenable for automatic processing by computer.

2. Universality: The parser should work for both positional and free word

order languages. Note that this is not the same as the notion of universality in modern generative linguistics.

3. Modularity: Modularity is desirable for reducing complexity and facilitating parallelism. More importantly, the very aim of UCSG is to divide and conquer. We aim to overcome complexity of syntax by dividing it into modules appropriately so that each module can use a type of grammar that is both necessary and (just) sufficient.

4. The structural descriptions produced by the parser must clearly depict linear, hierarchical and functional structures of the input sentences.

5. Computational efficiency is also high on priority.

3.2 Elements of UCSG Syntax

Syntax deals with the internal structure of sentences. A sentence is simply a sequence of words in its surface form. When a sentence is uttered in speech form, the words form a sequence in temporal order. When written, a sentence is a linear sequence of words. Sentences actually encode information, intentions, attitudes etc. Thus they form a bridge between the mental representations of the producer (speaker or writer) and the comprehender (listener or reader). There is a deeper, complex structure that is linearized when spoken or written down. The major goal of syntax is to explicate the underlying structure so that structure can be related to meaning.

Linear order is one of the most basic and most obvious elements of sentence structure. Words come in a sequence and changing the positions of words may change the meaning or render the sequence anomalous. *Ram killed Ravan* and *Ravan killed Ram* mean entirely different things. *The cat ate the rat* makes sense but *rat cat ate the* makes no sense. However, strict word order is not an absolute requirement in all cases. Meaning change between *I will read the book if I find time* and *If I find time I will read the book* is more subtle. In *A barking sound the Shepherd hears* and *The Shepherd hears a barking sound* who heard what has not changed although there is some change in focus. In Indian Languages, there is a great deal of flexibility in word order. Sanskrit is the ultimate - almost every permutation is valid and the basic meaning remains unchanged in all cases. The title song of the TV serial on the great epic *Mahabharata* reads *saarati jiske bhale* *sri krishna bharat partha ki* in Hindi and means the same as *bharat, jiske bhale sri krishna partha ki saarati.* The order of words has changed to 6-2-3-4-5-7-8-1. In fact this freedom in word order is what makes these languages so much more pleasing and poetic.

Linguists sometimes argue that there is an unmarked word order and syntax must primarily be concerned with this unmarked order. Word order changes can be accounted for by movement when it is a basic syntactic phenomenon and by topicalization etc. in other cases. Positing an unmarked word order is perfectly fine but trying to account for all the different word orders through movement would be circumlocutious, inefficient and inelegant. There is a great deal of freedom in word order which must be accepted as a plain fact in these *relatively free word order* languages and syntax must directly face this fact. Word order changes are not rare or extraordinary phenomena, they are regular and frequent. Any theory that is rooted in phrase structure rules and trees is inherently word order specific and there is no easy way of relaxing the word order constraints. $A \rightarrow B\ C$ is different from $A \rightarrow C\ B$ although both say that B and C are the constituents of A and A is composed of B and C. If all you have to say is B and C constitute A without implying any order, phrase structure rules are not well suited.

It is not the case that word order changes are without constraint even in the so called free word order languages. Very few languages have near total freedom of word order. In fact when we talk of movement what moves are not individual words but whole groups of words. For example, we can say *A barking sound the shepherd hears* or *The shepherd hears a barking sound* but not any other sequence where the groups of words *The shepherd* and *A barking sound* are jumbled or mixed up in any way. Such groups of words can be easily identified in all languages. When they move, they move as a whole. Such groups of words can often be given out as answers to questions. They may be translated as units. We shall see that such groups of words can be treated as atomic units in higher levels of syntactic description and we shall also see the merits of doing so. Word groups form an appropriate level of linguistic description. Syntax must deal directly with word groups. Higher levels of processing must be in terms of word groups, not words.

It may be noted that traditionally word groups as we have described above are also called phrases. We have intentionally avoided the term *phrase* as this term has already acquired very different connotations and usage. By using different terminology we hope to reduce possible confusions. We shall define the term *word group* after we characterize its properties in more detail later.

Word groups can be classified into verb groups, noun groups, adjective groups and so on based on the essence of the meaning as indicated by the *head* of the word group. Note that a verb group is very different from the usual notion of a verb phrase in linguistic theories. A verb group is a group of words with a verb as its head. It conveys the connotation of some action or state. There is nothing more to it than this. Thus a verb group in English may include auxiliary verbs and a main verb. A verb phrase, on the hand, may stand for not only a verb group but also other word groups such as the object of the verb, complements, modifiers etc.

A word group is a level of linguistic description that lies in between the levels of words and sentences. There is another level of description that has somehow not received the serious direct treatment that it deserves. This is the level of *clauses*. A clause is intermediate in size between word groups and sentences. Although well known in traditional grammars, modern linguistic theories have mixed up phrases and clauses adding to a lot of confusion and unnecessary complexity. Clauses are important units of description and syntactic theories need to deal with clauses directly as independent and distinguishable units of linguistic description. We shall define clauses below. We shall also show the merits of dealing with the structure of sentences at the levels of word groups and clauses separately.

A clause is a verb group along with the associated noun groups. The verb group denotes some action or a state and the associated noun groups may denote the doer of the action, the experiencer of the action or state, modifiers of place, time, manner etc. In effect a clause represents a complete predicate. As such it is an extremely important unit of description. A sentence may consist of one or more related predicates. Thus sentences may be simple sentences or multi-clause sentences. The structure of a sentence can be understood in terms of the individual clauses and the inter-relationships between the various clauses.

It is possible to talk about a whole event and we do this by using a whole clause as the subject of a sentence or another clause. Similarly the object of a clause can be a whole clause. Relative clauses can be used to modify nouns. Thus clauses in a sentence tend to form complex hierarchical structures. Clauses in a sentence cannot be understood simply as linear sequences of words. Hierarchical nesting is an innate property at the level of clauses. Syntactic theories must have the tools to deal with these hierarchical, nested structures.

On the other hand, there are no hierarchical structures within word groups. Word groups are simple linear sequences of words. There is no need to analyze the nested structures within word groups at a syntactic level although nested structures are evident at a semantic level. The modifier-modified relationships are often nested. It is basic tenet of UCSG that such nested structures, which are largely semantic in nature, cannot be captured within the boundaries of syntax. Syntax cannot account for the nested semantic structures within word groups and there is no point trying to do so.

Thus word groups and clauses are very different in nature. Word groups are simple linear sequences of words and as such can be captured very effectively using very simple grammars. Clauses, on the other hand, show complex nested structures and will need the power of grammars that can capture these hierarchical structures. Separating the two is therefore a very good idea. Otherwise, we will be forced to use the more complex grammar to deal with even the simpler aspect.

Clauses may be used as modifiers of other word groups. For example, relative clauses can modify noun groups. There is a lot to gain by treating these clauses separately rather than considering the clauses as part of the word group being modified. Clauses are made up of word groups and as such they are at a higher, more complex level than word groups. How can smaller, simpler, lower level units include bigger, more complex, higher level units inside them? Most grammar formalisms have not made clear distinctions between these two levels and so they end up making the whole thing unnecessarily complex and inefficient. By appropriately dividing the problem into sub-problems and by devising the simplest grammars that are necessary and sufficient in each case, UCSG achieves all the three goals of modularity and simplicity of grammar, efficiency of parsing and, as we shall see below, universality.

Now we are ready to define word groups. In UCSG we define a *word group* or simply a *group* as "a structural unit, a non-overlapping and non-recursive sequence of words, that can as a whole play a role in some predication[93]". Every word group has a head which defines the type of the group. Thus we may talk of noun groups, verb groups, etc. Word groups are understood best in terms of answers to questions. Each word group answers a particular question such as who, where or when. Thus word groups are similar to *chunks* [4, 85], yet they are often more semantically oriented. Chunks in many chunking systems are a bit too superficial - a preposition may stand out as a chunk by itself, for example.

Apart from the linear and hierarchical structures captured using word groups and clauses respectively, sentences also depict functional structure in terms of functional or thematic roles played by various constituents and the functional inter-relationships among the various constituents. These functional aspects cannot be captured either by the linear structure of word groups or by the hierarchical structure of clauses. Thus there is need to posit a separate third component to describe the functional structure of sentences. The UCSG architecture described in the next section shows how these three aspects of structure can be captured in a simple, elegant, efficient and universal manner.

In summary, the structure of a sentence can be described in terms of the linear, hierarchical and functional structure. The UCSG architecture adapts a divide and conquer strategy and proposes the simplest and most efficient grammar for each of the modules. It also shows how the three modules can be inter-connected to achieve simplicity and efficiency without compromising on universality. The modular nature of UCSG architecture also makes grammar development simpler. In the next section we sketch the architecture of UCSG and describe its merits.

3.3 UCSG Architecture

We have seen that the structure of a sentence can be understood in terms of linear order of words in word groups, hierarchical or nested structure of clauses and the functional roles taken by the various word groups in each of the clauses in the sentence. Here we describe the overall architecture of UCSG syntax and show how each of these aspects can be elegantly and efficiently handled. The inter-relationships between the three aspects of structure will also become evident.

By definition, functional or thematic roles are assigned to whole word groups. Word groups are treated as atomic and indivisible units for this purpose. We will not need to look at individual words at all. As we shall see below, we will not need to look at individual words even to analyze the structure of clauses. Therefore it is appropriate to recognize the word groups in a given sentence first and thereafter treat the sentence as a sequence of word groups rather than as a sequence of individual words. This way the effective length of the sentence is reduced and grammars become simpler and analysis becomes more efficient. For example, if the average length of a word group is three words and a particular parsing algorithm is of cubic time complexity, we get a 27-fold speed up.

By definition, every clause has exactly one verb group. Every clause will also have other roles such as subject and object. In other words, every clause has its own functional structure. We will need to analyze the functional structure of each of the clauses in a multi-clause sentence. Most grammar formalisms have no mechanism to isolate the various clauses and therefore every noun group is a potential subject or object of every verb group. The problem of assignment of thematic roles to word groups as posed here is exponential in complexity and the complexity can only be reduced to a limited extent through the application of various constraints. For example, in English the subject normally precedes the verb. In Indian languages, which are free word order languages, the subjects and objects can be identified by the case as indicated by morphology. On the other hand, if the structure of clauses in a sentence could be analyzed independently and the various clauses isolated, then it will become possible to define the functional structure independently for each of the clauses. Grammars become so much simpler and parsing so much more efficient. Thematic role assignment could proceed clause by clause and only the word groups inside the clause will need to be considered. Word groups outside the clause boundaries need not even be considered - there is no need to eliminate these through other constraints. This is possible because as a general principle, word groups associated with a particular clause do not cross the clause boundaries. The problem of role assignment becomes inherently less complex. In a sentence with c clauses where each clause has r roles to be assigned to r word groups, this clause by clause analysis reduces the complexity from $(c * r)!$ to $c * (r!)$ - an enormous gain.

In UCSG we show that it is in fact possible to analyze the hierarchical structure of clauses and also determine the clause boundaries without resorting to any of the functional level constraints. The number and type of clauses, their inter-relationships, as also the clause boundaries can be determined before and without applying functional level constraints such as word order (subject comes before verb - for English) or case (subject is in nominative case - for Indian languages). A clause structure analyzer before and without functional structure analysis is the most significant feature of UCSG.

The figure 3.1 summarizes the inter-relationships between the three modules of UCSG, each dealing with one the three basic aspects of sentence structure.

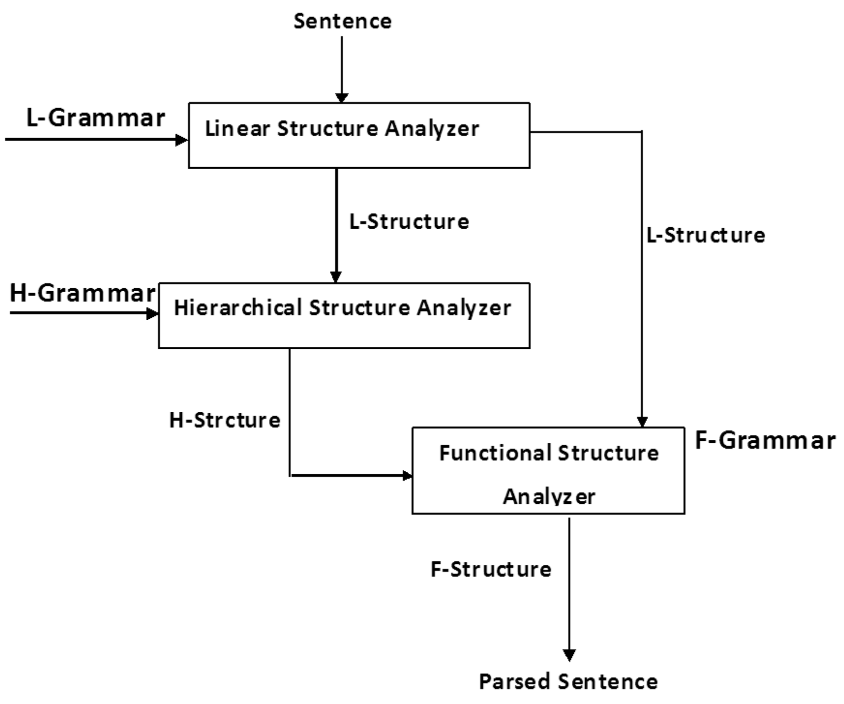


Figure 3.1: UCSG Full Parsing Framework

1. L-Module: The L-Module (Linear Structure analyzer) takes a sentence as

input and produces its L-Structure as output. The main task of the linear structure analyzer is to identify all potential word groups in the given sentence. L-module can be viewed as a chunker or a shallow parser. The linear structure analyzer obtains words in the sentence, looks up the lexicon and carries out morphological analysis where required. A POS tagger can be included. It then identifies all potential word groups. L-Module obtains all potential word groups in a sentence in a single left to right scan of the sentence in linear time using Finite State Machines. Finite state machines are both necessary and sufficient for identifying all potential word groups in a given sentence in any language. It is wasteful and unwise to use any more complex type of grammar. Finite state grammars are far simpler and more efficient than phrase structure grammars of the context free type.

L-Grammar is a regular grammar. With the recent additions, the extended L-Grammar is shown in tables of chapter 4.

2. H-Module: The task of H-Module is to produce the hierarchical structure of the clauses in a sentence.

   UCSG make a distinction between three kinds of clauses denoted f_clause, rel_clause and sub_clause. Every sentence must include one and only one f_clause apart from those embedded within the rel_clauses and sub_clauses if any. rel_clauses indicate relative clauses - these clauses modify a participant role in the F-Structure rather than directly fill one such role. sub_clauses fill participant roles. Thus clausal subjects, clausal objects and the clausal modifiers of place, time etc. are all termed as sub_clauses.

   By definition, every clause has one verb group in it and this must definitely be part of that clause. UCSG determines the other constituents belongs to a clause by the observation that one of the boundaries of every clause in a sentence is overtly marked by certain kinds of words or morphemes called sentinels. For example, relative clauses begin with relative words like 'who', 'which' and 'that'. The complementizer 'that' is another important sentinel. The beginning of relative clauses is marked by the relative word sentinel 'rl' and the beginning of subclauses is marked by a subordinate conjunction sentinel 'sb'. Verb groups and sentinels are necessary and sufficient to identify clauses and also partly determine the clause boundaries and inter-clause relationships.

In English, the end of the matrix clause and the start of other clauses are well marked by sentinels. There are very strong constraints on the sequences of verb groups and sentinels. Every clause includes one verb group and one sentinel. Verb groups and sentinels are therefore like left and right parentheses and the rules of hierarchical structure enforce proper nesting of these parentheses. Thus context free grammars are both necessary and sufficient. It is wasteful and unwise to use any more complex type of grammar.

Indian languages are handled by a very similar and parallel set of grammar rules. The differences are parametric - Indian languages are SOV languages and sentinels come at the right end of the clauses, rather than at the left end. Everything else remains the same, thus exhibiting the universal nature of syntax.

Hence H-Module efficiently determines the hierarchical structure of the clauses by looking at only a few constituents in the given sentence. For each clause, one of its boundaries is found deterministically and for the other boundary, two limiting positions are found. Having partially analyzed the hierarchical structure of clauses we can localize our search to the individual clauses for the assignment of functional roles to the arguments and modifiers.

In summary H-Module takes a string of verb groups and sentinels as input from the L-Module and produces all possible clause hierarchies as output by using the H-Grammar. Only a few simple phrase structure rules are required.

3. F-Module: In this last and final module, functional roles are assigned to the various participants in each of the clauses in the input sentence. USCG posits a set of eight basic functional roles, keeping the question answering paradigm in view: subject, object, subject qualifier, object2, space, time, adverbs and clause link. Since the clause structure would have already

been analyzed, the functional structure analyzer works clause by clause. Since lower level clauses play specific roles in the higher level clauses in which they are nested, starting with the matrix clauses and working down from the hierarchy would greatly facilitate role assignment. Also, since the inter-clause dependencies including the sharing and displacement of constituents are all related to the hierarchical structure of clauses, working top-down and passing down information about missing, displaced and shared constituents will make the functional structure analysis completely a clause internal problem, thereby getting rid of all problems of long distance dependencies. A combination of top-down and bottom-up strategies is employed for role assignment. Sub categorization frames and selectional restrictions provide the top-down constraints while the surface case marking information attached to the word groups from the bottom-up constraints. Any F-Structure that satisfies all these constraints would be a valid result. In this process, exact clause boundaries would also get determined. Thus UCSG works from whole to part, not left to right or right to left.

3.4 Advantages of UCSG Architecture

3.4.1 Modularity

UCSG divides the task of syntactic analysis into three major modules and proposes the simplest and most efficient types of grammar for each. This neat modularity leads to simple grammars and efficient parsing. This also leads to a high degree of language independence - only a few simple parametric variations are required to handle widely varying language families including English and Indian languages. Also, it is easy to incorporate additional modules such as a POS tagger. Lastly and most importantly, this modular architecture facilitates incorporation of statistical techniques into the architecture.

3.4.2 Computational Complexity

A Type-3 grammar is used to recognize all possible word groups in a single scan of the given sentence in linear time. All further work is done in terms of word groups, not individual words, thereby reducing the effective data size. Clause structure is analyzed using type-2 grammars in cubic time and only a small number of grammar rules are required. The input to clause structure analyzer is just the sequence of verb groups and sentinels, again reducing the input data size considerably. Finally, thematic roles are assigned one clause at a time, working from whole-to-part starting from the matrix clause. Role assignment is a factorial problem and taking out C, the number of clauses in the sentence from the factorial term is a great improvement. These features make UCSG computationally highly efficient.

3.4.3 Functional Structure

Unlike other recent parsing systems such as Collins parser and Charniak parser, UCSG produces thematic role assignments that are more semantic in nature and well suited for applications such as information extraction and question answering.

3.4.4 Universality

UCSG can naturally handle both positional languages like English and relatively free word order languages like the Indian languages under an equal footing[94].

3.4.5 Long Distance Dependencies

UCSG analyzes the hierarchical structure of clauses before and without applying any of the functional structure constraints. Functional structure itself is analyzed clause by clause working down the clause structure tree, starting from the root (the matrix clause) and working towards more and more deeply embedded clauses. This eliminates the problems related to long distance dependencies. Dependencies are not related to distance, bit to clause structure. Once the clause structure is determines, all dependencies become local to the respective clauses. This is a very significant contribution of UCSG.

3.5 Drawbacks of the original UCSG Full Parsing System

- No claims were made about the psychological reality of the proposed architecture. Although motivated by deeper semantics, the processing itself was mostly based on surface syntax. The aim was to do more by doing less.

- Through evaluations on large scale data had not been done, either at the level of individual modules or as a whole.
- UCSG was almost entirely based on knowledge based or linguistic approach. Hardly any statistics was used although the architecture itself facilitated inclusion of statistical techniques.

Chapter 4

UCSG Shallow Parsing Architecture

4.1 The Design of the UCSG Shallow Parsing Architecture

The main contributions of this thesis are:

1. A Shallow Parsing Architecture that enables the development of wide coverage shallow parsing systems starting from a large POS tagged corpus by a judicious combination of linguistic and statistical approaches.

   This thesis shows that Finite State Grammars with very high Recall can be built. Chunk level HMMs can be developed from a large POS Tagged corpus using the Finite State Grammar-Parser and these HMMs can in turn be used for rating and ranking the chunks produced by the Finite State parser. Best first search strategy can be used to produce appropriate chunk sequences (parses) in ranked order. This architecture also shows how a bootstrapping strategy can be used to improve the HMM parameters and hence the performance of the whole parser.

2. A wide coverage shallow parsing system for English is developed using this architecture to substantiate the claims made.

The details of the UCSG English parser and all the experiments conducted and results obtained are given in the next chapter. In this chapter we develop the UCSG Shallow Parsing Architecture.

UCSG shallow parsing architecture is set within the UCSG full parsing architecture described in chapter 3. Here, the focus is on chunking - identifying chunks or word groups, handling ambiguities, and producing parses (chunk sequences) for given sentences. This can be extended to include thematic role assignment and clause structure analysis leading towards a full parser.

Purely linguistic approaches have not proved practicable for developing wide coverage grammars and purely statistical or machine learning approaches are also impracticable in most cases due to the non-availability of large enough parsed training corpora. Only a judicious combination of the two approaches can perhaps led to wide coverage grammars and robust parsing systems. UCSG shallow parsing architecture proposes one such solution [75].

Words in natural languages exhibit lexical ambiguities - many words have more than one POS tag. Dictionaries simply list possible POS tags for each word without giving any idea as to which POS tag is most appropriate for a given word in a given context. Also, sentences in natural languages exhibit rich and varied structures and abound in structural ambiguities as well. The problem of finding the correct structure for a given sentence is of exponential complexity. Of these large number of possible combinations, a parser is expected to produce the *only* correct combination, ruling out all other invalid combinations. Natural languages are ambiguous at higher levels too and oftentimes there are more than one syntactically valid solution. A parser is expected to produce *all* syntactically valid combinations at every level. Meeting these two requirements of accepting all valid structures and rejecting all invalid structures at the same time is extremely difficult. If we try to make the grammar a bit more general to accept some cases that were being rejected, we find that some other invalid structures are also getting accepted. If we try to make the grammar more restricted, we find that some valid structures are also getting rejected. Although human languages are largely rule governed, the world is not as simple and neat as we may wish it to be and practically developing a grammar that general enough to accept all valid structures and restricted enough to reject all invalid structures is not at all easy. In fact experience shows that nobody has been able to develop a wide coverage grammar for any human language in the world so far which satisfies both the *all* and *only* requirements.

It therefore makes practical sense to take up the two requirements one by one. It is easier to develop a grammar that is somewhat over-general by design so that all valid structures can be captured. Once all the valid structures are captured, we may then impose filters to remove invalid combinations. Alternatively, we may include statistical components to rate and rank the structures produced so that the correct ones can be preferred over the invalid ones. If we instead start with a highly restricted grammar, extending it and making it more general will not be easy. In the UCSG shallow parsing architecture, we propose a linguistically motivated grammar-parser to capture all possible word groups in a given sentence and then employ a statistical component for rating and ranking the chunks so produced. We will see that this combination is practicable and enables the development of wide coverage and robust shallow parsing systems.

Sentences in natural languages exhibit both dominance relations and dependency relations. Dominance is expressed in terms of the constituent structure, usually in the form of a tree structure. Tree structures depict parent-child, part-whole, modifier-modified, recursive and other such kinds of hierarchical, nested structures. They also depict linear structure in terms of the linear position or word order. This may look very natural and intuitive for languages that enforce a strict word order but this is highly unnatural and undesirable for languages where order of words in a sentence is relatively or completely unimportant or irrelevant. Although un-ordered trees are conceivable, the phrase structure rules that are often used to generate such tree structures cannot shed their inherent ordered nature. UCSG argues that phrase structure rules (or equivalents such as trees) are not suitable for all aspects of syntax. Trees are suitable where there is a combination of linear and hierarchical structures. If any one of these is not relevant, trees and phrase structures rules should not be used. Trees and phrase structure rules are appropriate and essential for dealing with the nested structure of clauses in a given sentence but not for recognizing chunks.

Chunks by definition are non-recursive and non-overlapping. Linear order of words within a chunk is always important. This is true of positional languages such as English and relatively free word order languages such as Indian languages. However, there is no hierarchical structure within chunks and type-2 grammars and trees are not required. The simpler, more efficient regular grammars (or equivalents such as regular expressions and finite state machines) are sufficient. In UCSG we show that finite state machines are sufficient to recognize all possible word groups in any given sentence. Word groups are simple ordered lists of words and there is no nested, hierarchical or recursive structure.

There are of course nested structures within word groups but we argue that capturing these hierarchical structures is beyond syntax. Consider the example:

Water Meter Cover Adjustment Screw

We understand this as the adjustment-screw on the cover of the water-meter. We understand water-meter as a meter used for measuring the flow of water. We understand that water-pump is a pump used for pumping water and we understand that cast-iron-pump is a pump made up of cast-iron, not a pump that pumps cast-iron! How do we understand all this? It is clear that while the modifier-modified relations within chunks show up as some kind of hierarchical or nested structure, capturing them is beyond syntax - a high degree of semantics and world knowledge is called for. While one may argue that there is hierarchical structure within chunks, it is beyond the scope of a purely syntactic system to capture such relations and in chunking we ignore all hierarchical, nested, recursive structure and treat chunks simply as ordered sequences of words.

Note also that many times linguists use recursion to implement mere repetition. Repetition of adjectives within noun groups does not call for any recursion, although capturing the modifier-modified relations is nested by nature. In UCSG we show that high performance finite state grammars can be developed to accept all valid chunks in a given sentence.

There is another source of confusion between linear and hierarchical structure. Noun groups may be modified by relative clauses. It has been traditional to consider the relative clause as part of the noun phrase. Thus noun phrases can have sentences within them and sentences of course have noun phrases within them. There is thus a recursive dependency between noun phrases and sentences. Other grammar formalisms have used recursive phrase structure rules for capturing this dependency. As a result, they lose the distinction between phrases and clauses. Semantically the relative clause *is* a part of the noun phrase. However, we should make a distinction between the structure of word groups and the structure of clauses. At chunking level, there are no nested or recursive structures.

Order of words and the corresponding POS tags within chunks is significant and the finite state grammars capture exactly this. In UCSG we view chunks from a probabilistic point of view as well. Not all words can start a chunk, not all words appear next to another given word with equal probability. Linguistics deals only with possibilities - all valid structures are possible and all invalid structures are impossible. This is however, not merely a binary, yes-or-no question. There are degrees of possibilities. All grammatically valid chunks have already been obtained using a finite state grammar but not all of these are equally probable. In UCSG we view chunks as Markov processes and model chunks in terms of probabilities of particular POS tags starting a chunk of a given type, probabilities of transitions within chunks from one POS tag to another, and the probabilities associated with specific words taking on specific POS tags. We thus find it natural to use Hidden Markov Models (HMMs) to capture the statistical nature of chunks in a given language.

Since we already know the chunks, we need not use HMMs to obtain chunks from the given sentence. All we need to do is to evaluate the chunks from a probabilistic point of view and rate and rank the chunks. Thus we do not need Viterbi search and only the simpler and more efficient evaluation algorithm is required. Note that we are not pruning, we are only sorting the chunks based on probabilities.

HMMs rate and rank the competing chunks at every position within a given sentence, thereby giving us a probabilistic leverage to chose the *best* chunks at each position. However, the locally best chunks at any given position within a sentence do not necessarily add up to give us the best overall chunk sequence for the whole sequence. This is because HMMs have only considered within-chunk constraints and across-chunk information and global linguistic constraints have not been used as yet. We therefore propose a third module to perform some kind of best first search over all the possible chunk sequences so that all possible chunk sequences can be produced in best first order. In principle, we need not impose any pruning even here and this guarantees that the correct parse (chunk sequence) will always be produced (although not necessarily at the top) as long as the finite state grammar had identified all the correct chunks. Thus the performance of the whole shallow parser in terms of producing correct chunks sequences is limited only by the performance of the finite state grammar. We show in the next chapter that very high performance can actually be achieved.

Figure 4.1 shows the basic UCSG Shallow Parsing Architecture:

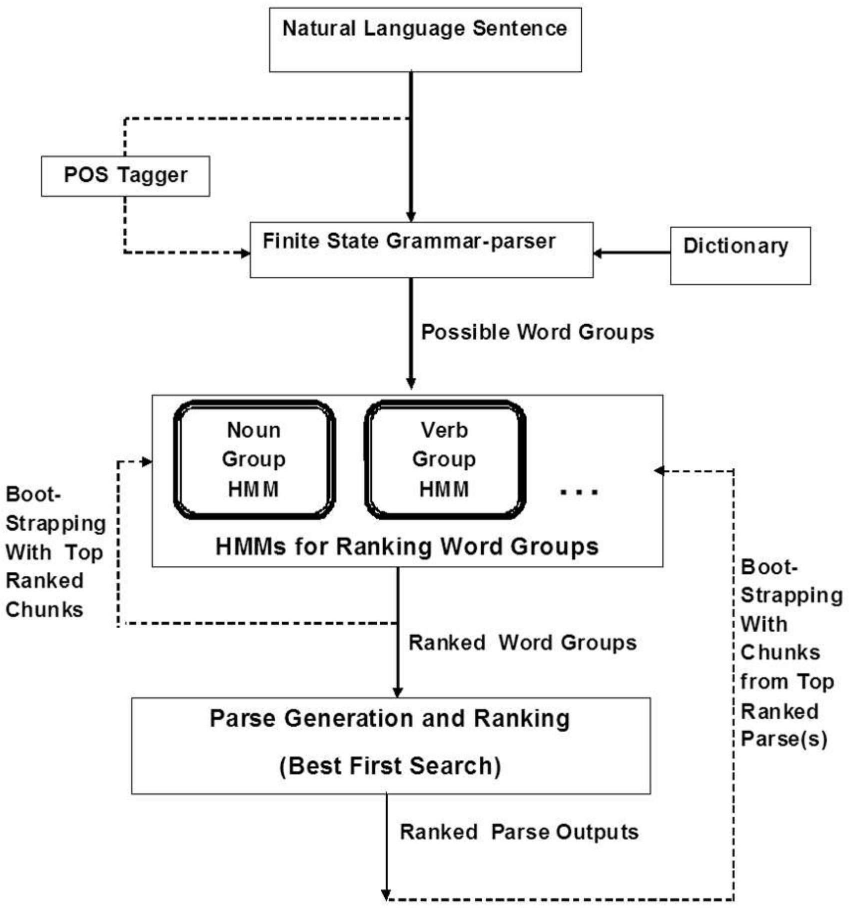

Natural Language Sentence
POS Tagger
Finite State Grammar-parser
Dictionary
Possible Word Groups
Noun Group HMM
Verb Group HMM
. . .
HMMs for Ranking Word Groups
Boot-Strapping With Top Ranked Chunks
Boot-Strapping With Chunks from Top Ranked Parse(s)
Ranked Word Groups
Parse Generation and Ranking
(Best First Search)
Ranked Parse Outputs

Figure 4.1: UCSG Shallow Parsing Architecture

The input to the parsing system is one sentence, either plain or POS tagged. Output is an ordered set of parses. The aim is to produce all possible parses in ranked order hoping to get the best parse to the top. In this work, by parse we mean a sequence of chunks. Chunks are sequences of words.

A chunk or a "word group" as we prefer to call it in UCSG, is "a structural unit, a non-overlapping and non-recursive sequence of words, that can as a whole, play a role in some predication [93]".

Note that word groups do not include clauses (relative clauses, for example) or whole sentences. Every word group has a head which defines the type of the group. Word groups can be classified into verb groups, noun groups, adjective groups and so on based on the essence of the meaning as indicated by the *head* of the word group. Thus word groups are similar to *chunks* [91, 124]. Our word groups are also very similar to the *phrases* defined in the work of Beata Megyesi [85]. It may be noted that the terms *chunk* and *phrase* have been used in substantially different connotations elsewhere in literature. The word groups we produce in UCSG are hopefully closer to ideal, semantically oriented units of full parsing, as can be seen from the examples given at the end.

In our UCSG syntax, the definition of a chunk is motivated by question-answering perspective. Consider

Sentence: I am studying at University of Hyderabad.

```
UCSG outputs the following word groups

<ng>[<PNN><i>]</ng>

<vg>[<VBB><am> <VVG><studying>]</vg>

<ng>[<PRP><at> <NN1><university> <PRF><of> <NP0><hyderabad>]</ng>
```

The word groups produced can thus be viewed in terms of answers to basic questions such as who, whom, where, when etc. For example, if you ask a question “where are you studying”, the answer is “at University of Hyderabad”. Observe that many chunking systems in the world today treat prepositions as chunks in their own right. Some chunkers break ‘University of Hyderabad’ into two chunks. See examples below:

Memory based shallow parser [32, 86] gives the following output:

```
[NP I/PRP NP]
```

```
[VP am/VBP studying/VBG VP]

{PNP [Prep at/IN Prep] [NP University/NNP NP] PNP}

[Prep of/IN Prep]

Hyderabad//VBD ./.
```

Note that the word Hyderabad is not part of any chunk.

CCG shallow parser [27] gives the following chunks:

```
[NP I]

[VP am studying]

[PP at]

[PP of]

[NP Hyderabad]
```

The word “University” is missing altogether.

Thus our word groups are a bit more semantically oriented and as such, more suitable for deep parsing as also for various NLP applications. We have set for ourselves a more challenging task and our results must be viewed keeping this in mind.

4.2 The Technology of the UCSG Shallow Parsing Architecture

4.2.1 Finite State Grammar-Parser

Only linear order, repetition and optional items are relevant for recognizing chunks - there are no nested or recursive structures to consider. Finite state grammars efficiently capture linear precedence, repetition and optional occurrence of words in word groups but not arbitrarily deep hierarchical nestings or general dependencies across constituents. Finite state machines are both necessary and sufficient for recognizing word groups [93]. It is also well known that finite state machines are computationally efficient - linear time algorithms exist for recognizing word groups. Finite state grammars are also conceptually simple and easy to develop and test. It may be repeated that detailed analysis of the internal structure of word groups (modifier-modified relationships, for example) is beyond the scope of the current system.

The Finite State module accepts a sentence (either already POS tagged or tagged with all possible categories using the dictionary) and produces an unordered set of possible chunks taking into account all lexical ambiguities.

4.2.1.1 Finite State Parser

During linear structure analysis all potential groups in a given sentence are to be recognized. Linear structure analysis takes care of lexical ambiguities and groups may overlap one another. The following algorithm identifies all potential word groups in a given sentence in a single left-to-right scan. This algorithm works for both deterministic and nondeterministic state transition diagrams. It simulates parallel processing. Instead of maintaining a single current state it maintains a current_state_set. Each word in the input sentence is considered only once and an amount of time bounded by the size of the grammar is spent per word. Hence the algorithm is linear in time complexity.

Pseudo Code for Linear Structure Analysis:

```
MAIN()
        initial_state_set := [ ]
```

```
    for each of the initial states $s_i$ in the network do
        initial_state_set := union([($s_i$,‘ ’)],initial_state_set)
        current_state_set := initial_state_set
        step through the words in the given sentence and for each word w
            advance(current_state_set,w)


ADVANCE(current_state_set,w)
    new_state_set := [ ]
    for each state (s,str) in current_state_set do
        for each out going arc a do
            if **any** of the categories of w matches the arc a
              begin
                  new_state_set := union( [(end_state(a),concat(str,w))]
                                          ,new_state_set)
                  if terminal_state(end_state(a)) then
                  begin
                      output(concat(str,w))
                      new_state_set := union(new_state_set,initial_state_set)
                  end
            end
    current_state_set := new_state_set
```

4.2.2 HMMs for Rating and Ranking Chunks

The second module is a set of Hidden Markov Models (HMMs) used for rating and ranking the word groups produced by the Finite State Grammar. The hope is to get the best chunks near the top. This way, although we are not restricting chunk generation to *only* the appropriate chunks in context, we can hope to get the right chunks near the top and push down others.

Words are observation symbols and POS tags are states in our HMMs. Formally, a HMM model $\lambda = (\pi, A, B)$ for a given chunk type can be described as follows:

Number of States (N) = number of relevant Categories

Number of Observation Symbols (M) = number of Words of relevant categories in the language

The initial state probability

$$\pi_i = P\{q_1 = i\} \tag{4.1}$$

where $1 \leq i \leq N$, q_1 is a category (state) starting a particular word group type.

State transition probability

$$a_{ij} = P\{q_{t+1} = j | q_t = i\} \tag{4.2}$$

where $1 \leq i, j \leq N$ and q_t denotes the category at time t and q_{t+1} denotes the category at time t+1.

Observation or emission probability

$$b_j(k) = P\{o_t = v_k | q_t = j\} \tag{4.3}$$

where $1 \leq j \leq N$, $1 \leq k \leq M$ and v_k denotes the k^{th} word, and q_t the current state.

While building HMMs, a manually checked and certified chunked corpus can be used if available. In this case, HMM parameters can be estimated right away. However, such labelled training data is rarely available. When no parsed corpus is available, we can rely on a POS-tagged corpus. In the latter case, a bootstrapping strategy is proposed to refine the HMM parameters later. See figure 4.2. We first pass a large POS tagged corpus through the Finite State module and obtain all possible chunks. Taking these chunks to be equiprobable, we estimate the HMM parameters by taking the ratios of frequency counts. One HMM is developed for each major category of chunks, say, one for noun-groups, one for verb-groups, and so on. The B matrix values are estimated from a dictionary that includes frequency counts for each word in every possible category.

We simply estimate the probability of each chunk using the following equation :

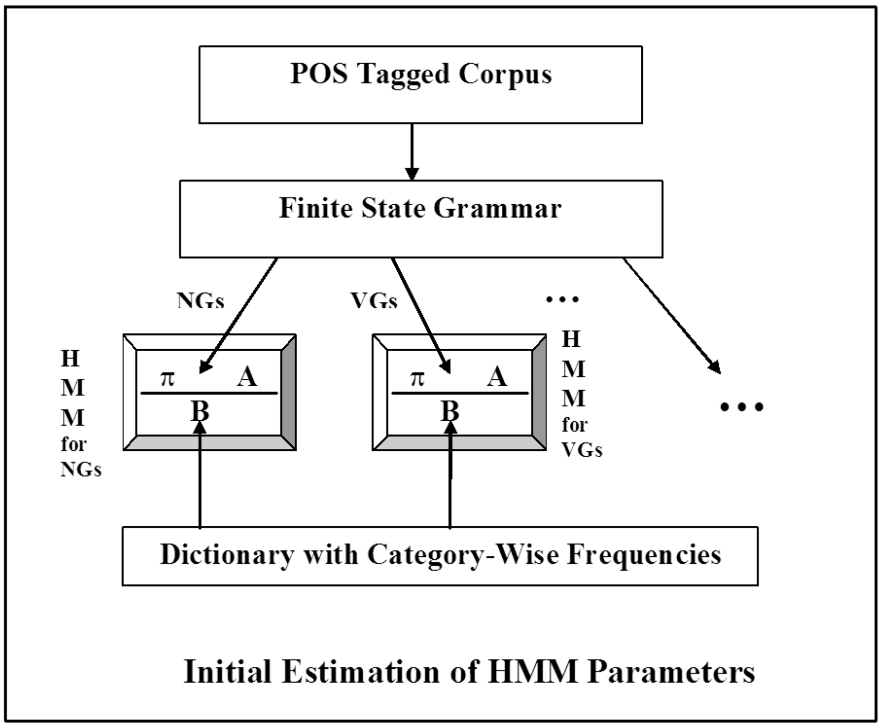

POS Tagged Corpus
Finite State Grammar
NGs
VGs
...
HMM for NGs
HMM for VGs
π A
B
...
Dictionary with Category-Wise Frequencies
Initial Estimation of HMM Parameters

Figure 4.2: Initial Estimation of HMMs

$$P(O,Q|\lambda) = \pi_{q_1} b_{q_1}(o_1) a_{q_1,q_2} b_{q_2}(o_2) a_{q_2,q_3} \cdots a_{q_{t-1},q_t} b_{q_t}(o_t) \tag{4.4}$$

where q_1 ,q_2, $\cdots$, q_t is a state sequence, o_1 , o_2,$\cdots$, o_t is an observation sequence. Note that no Viterbi search involved here and the state sequence is also known. Thus even Forward/Backward algorithm is not required and rating the chunks is therefore computationally efficient.

The aim here is to assign the highest rank for the correct chunk and to push down other chunks. Since a final parse is a sequence of chunks that covers the given sentence with no overlaps or gaps, we evaluate the alternatives at each position in the sentence in a left-to-right manner.

Here, we use *Mean Rank Score* to evaluate the performance of the HMMs. **Mean Rank Score is the mean of the distribution of ranks of correct chunks produced for a given training corpus.** Ideally, all correct chunks would be at the top and hence the score would be 1. The aim is to get a Mean Rank Score as close to 1 as possible.

4.2.3 Parse Generation and Ranking

The third module is for identifying the best chunk sequence or global parse for a given sentence. This module generates all possible parses, hopefully in best first order. We can of course limit the number of parses generated if required but the ability to produce all possible parses is fundamental to the architecture. Note that we do not produces all possible parses first and then rate and rank them - the parse generation process inherently incorporates best-first search.

Choosing the locally best chunks at each position in a given sentence does not necessarily give us the best parse (chunk sequence) in all cases. The HMMs are local to chunks and global information such as the probability of a chunk of a given type starting a sentence or the probability of a chunk of a particular type occurring next to a chunk of a given type are useful. These probabilities can be obtained from a fairly small chunked corpus. We have used best first search algorithm to get the best parse (chunk sequence) for a given sentence.

4.2.3.1 Best First Search Algorithm

In this section, we map our parse selection problem into a graph search problem and show how best first search algorithm can be used to find the best first parse.

Words and chunks in a sentence are referred to in terms of the positions they occupy in the sentence. Positions are marked between words, starting from zero to the left of the first word. The very first word is between positions 0 and 1. A word group containing the third and fourth words in the sentence can be referred as $W_{2,4}$.

The following steps describe how we map a given sentence and word groups present in the sentence into a graph.

- The positions in the sentence are treated as nodes of the resulting graph. If a sentence contains N words then the graph contains $N + 1$ nodes corresponding to the $N + 1$ positions in the sentence.
- Word group $W_{i,j}$ is represented as an edge form node i to node j.

- The probability of a word group $W_{i,j}$ given by HMM module and the transition probability from previous word group type to current word group type are combined to estimate the cost of an arc between the nodes i and j.
- We always start from the initial node 0. Length of the sentence N is the goal node.

Now our parse selection problem of a sentence containing N words becomes the task of finding an optimal path from node 0 to node N.

Pseudo Code for Best First Search Algorithm:

```
start_node = 0
goal_node = N #(length of the sentence)
cur_best = $< 0, 0, , , >$ # $< pos, prob, chunktype, path, parse >$
open_set = $\emptyset$
for i = 1 to k do
    repeat
        open_set = add_successor (cur_best, open_set)
        cur_best = find_best (open_set)
    until (cur_best.pos = goal_node)
    open_set = open_set $-$ cur_best
    print cur_best
done
function add_successor(cur_best, open_set)
chunkset = $\{x | x \in$ CHUNKS and x.from = cur_best.pos$\}$
foreach (chunkset) do
    elem.pos = chunkset[i].to
    elem.prob = cur_best.prob + chunkset[i].prob +
                    P (cur_best.chunktype, chunkset[i].type)
    elem.chunktype = chunkset[i].type
    elem.path = update (cur_best.path, chunkset[i])
    elem.parse = update_parse (cur_best.parse, chunkset[i])
    open_set = open_set $\cup$ elem
done
```

$open\_set = open\_set - \ cur\_best$

In best first search, we can inspect all the currently-available nodes, and rank them on the basis of our partial knowledge. Here high rank means that the node looks most promising in relation to the goal. At each step, we select the most promising of the nodes we have generated so far. We then expand the chosen node to generate it successors. If one of them is a solution, we can quit. If not, all those new nodes are added to the set of nodes generated so far. Again the most promising node is selected and the process continues. In the worst case, the best first search algorithm runs in exponential time because it expands many nodes at each level. In big-O notation, this is stated as $O(b^m)$, where b is the branching factor (i.e., the average number of nodes added to the open list at each level), and m is the maximum length of any path in the search space. Memory consumption is also a big problem, apart from time complexity. The number of nodes that are stored in memory rapidly increases as the search moves deeper into the graph and expanding too many nodes can cause the algorithm to run out of memory.

Beam search is a heuristic search algorithm that is an optimization over best-first search. Like best-first search, it uses a heuristic function to estimate the promise of each node it examines. Beam search, however, only unfolds the first m most promising nodes at each depth, where m is a fixed number, the "beam width". While beam search is space-bounded as a function of m, it is neither optimal nor complete when m is finite. As m increases, beam search approaches best-first search in complexity.

It is of course possible to incorporate a wide variety of other statistical and machine learning techniques for optimum chunk sequence selection. We would need a reasonable sized high quality chunked corpus for training. We have also explored A* best first search strategy. Linguistic constraints should be expected to play an important role in parse generation and ranking.

4.2.4 Bootstrapping

The HMM parameters can be refined through bootstrapping. Since we need to work with large data sets running into many hundreds of thousands of sentences, Baum-Welch parameter re-estimation would not be very practical. Instead, we can use parsed outputs to re-build HMMs. It may be recalled that originally HMMs were built from chunks obtained from the over-general finite state parser taking all chunks as equi-probable. By parsing a given sentence using the system and taking the top few parses only as training data, we can re-build HMMs that will hopefully be better. We can also simply use the top-ranked chunks for re-building the HMMs. This would reduce the proportion of invalid chunks in the training data and hence hopefully result in better HMM parameters. In the next chapter we shall see that this idea works and we can improve HMM parameters and improve parser performance as well.

4.3 Summary

We have proposed an architecture for wide coverage shallow parsing by combining finite state grammars and HMMs. Finite state grammars are simple, easy to understand and develop, and computationally efficient. HMMs can be built from a POS tagged corpus - there is no need for a large parsed training corpus to start with. In fact parsed corpora can be developed using the architecture proposed and these corpora can in turn be used to refine HMM parameters by bootstrapping. We have also posited a best first search strategy so that we can obtain all possible chunk sequences in best first order. This makes the parse output more suitable for further deep parsing and for other NLP applications. Even the word groups we produce in UCSG are motivated by deeper, semantic constraints. These ideas and claims are substantiated with experimental work as detailed in the next chapter where we describe our efforts in building a wide coverage shallow parsing system for English. Good performance has been obtained although we have not used any large scale parsed corpus for training and even the linguistic knowledge used is very limited.

Chapter 5

Experiments and Results

In this chapter, we describe our experiments with the various modules of UCSG shallow parsing architecture. We also compare the results with other shallow parsers. All the experiments have been carried out on a desktop PC with Pentium Core 2 DUO 1.86 GHz Processor and 1 GB RAM. The entire system has been implemented in Perl under Linux.

5.1 Building a Lexicon from a Large Corpus

Electronic dictionaries play vital role in any natural language processing system. Electronic dictionaries form an integral component of almost every activity in computational linguistics and NLP - word processing systems, spelling error detection and correction, grammar checking, office automation, morphological analysis and synthesis, parsing and generation, machine translation, question answering systems etc. Traditionally a dictionary or lexicon is intended for providing a valid list of words in a language, their meanings, pronunciations, etymologies etc. Dictionaries can be built for specific purposes and the contents and organization would vary accordingly.

Intelligent processing of natural language for real world applications requires lexicons which provide rich information about morphological, syntactic and semantic properties of words. Hence digital form of dictionary has considerable potential, especially if it can be built in a such way that it is compatible to the needs of various applications in language processing. One of the most important needs for a lexicon is natural language parsing. The dictionary used for syntax need not contain pronunciations, etymologies, meanings of words etc. The entries need to have their morpho-syntactic features such as gender, number, case, person and inflection. It is also very useful if the dictionary contains POS tags of the word along with the frequency of occurrence of words and also frequency of occurrence of word with a particular POS tag in a large corpus which covers different genres of a particular language .

Corpus-based lexicography is an effective way for building a dictionary for a languages, especially where word boundaries are explicitly marked. Corpora contain a lot of noise such as spelling errors, numbers, special symbols, expressions, tables etc. Proper nouns like names of people, organizations, places and other entities will occupy large percentage of a corpus naturally. Many words are likely to be less frequent and domain specific. Hence one needs to be careful in selecting words from a corpus to build lexicon.

The choice of corpus used for lexicon extraction plays a vital role in making the lexicon more robust. The corpus has to be large enough and it should cover different genres of text for a given language so that the dictionary extracted would be more robust and give good coverage. Since the British National Corpus (BNC) is fairly large POS tagged corpus, which is designed to represent a wide range of modern British English of different domains, we have employed the BNC corpus for dictionary extraction.

British National Corpus (BNC) [16] is a 100 million word (POS-tagged corpus) collection of samples of written and spoken language from a wide range of sources. The BNC World Edition contains 4054 text documents and occupies (including SGML markup) about 1.5 GB. In total, it comprises just over 100 million orthographic words (to be precise, 100,467,090 words), but the number of w-units (POS-tagged items) are slightly less at 97,619,934.

The Corpus is designed to represent as wide a range of modern British English as possible. The written part (90%) includes, for example, extracts from regional and national newspapers, specialist periodicals and journals for all ages and interests, academic books and popular fiction, published and unpublished letters and memoranda, school and university essays, among many other kinds of text. The spoken part (10%) includes a large amount of unscripted informal conversation, reordered by volunteers selected from different age, region and social classes in a demographically balanced way, together with spoken language collected in all kinds of different contexts, ranging from formal business or government meetings to radio shows and phone-ins.

5.1.1 Steps Followed Extracting the Lexicon from the Corpus

The first step is preprocessing which includes the removal of text format marks, e.g. SGML tags, deletion of redundant space characters, tabs and other special symbols etc. The corpus files contain header and other SGML tags, along with the POS tagged units. We have extracted only POS-tagged word units from the corpus. While extracting, we have done case folding so that there will be no repetition of words in lower and upper case. Each distinct word is called a type and each occurrence of such a word is called a token. There are 652,423 types and 97,619,946 tokens in the BNC corpus.

Coverage analysis deals with the examination of how much of a corpus can be covered by a given set of types. We performed a type-token analysis and prepared a list of types sorted in decreasing order of frequency of occurrence. By thresholding on this list, we have selected the most frequent n words in the language, for any given value of n. We then explored what percentage of words in a corpus are found in the list so selected. Here we have performed self-coverage analysis - coverage analysis on the same corpus from which the words are extracted. See the graph shown in figure 5.1. The self-coverage analysis of BNC corpus is also shown in table 5.1.

From the table 5.1, we can observe that the most frequent 200,000 entries have covered 99.39% of entire BNC corpus. Other 452,423 types are having nearly one token for each type - very infrequent in deed.

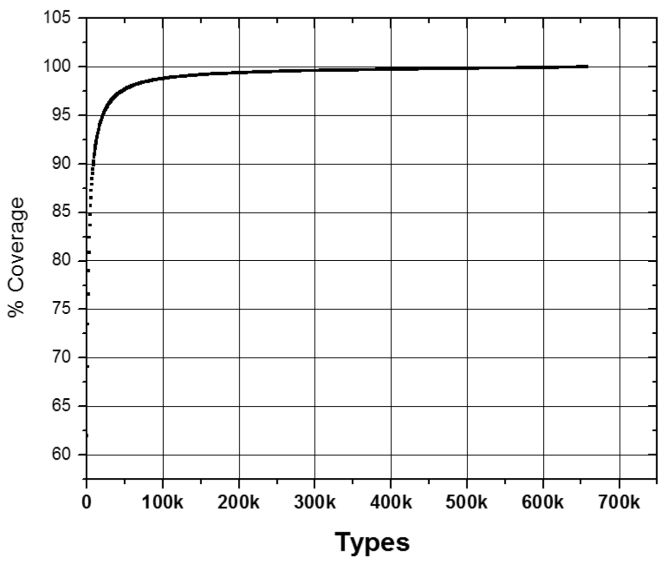


Figure 5.1: BNC Self Coverage Analysis

In this work, we have considered the first 200,000 types to build the dictionary. Even in these most frequent 200,000 types, there are cardinal numbers, special symbols, proper names and abbreviations etc. In the following steps, we have separated out these types to form the dictionary because it is very inefficient to keep them in the dictionary and also we have easy ways to handle them without having them in the dictionary.

5.1.1.1 Disambiguation of Ambiguous POS Tags

In English, many words are ambiguous, taking more than one potential tags. BNC is an automatically tagged corpus, tagged using the CLAWS POS tagger, which was developed by Roger Garside and his co-workers at Lancaster. CLAWS is a hybrid tagger, employing a mixture of probabilistic and non-probabilistic techniques. There are also 30 "Ambiguity Tags" used for tagging wherever the difference in probabilities assigned by the CLAWS automatic tagger to its first and second choice tags were considered too low for reliable disambiguation. For example, the ambiguity tag AJ0-AV0 indicates that the choice between adjective (AJ0) and adverb (AV0) is left open, although the tagger has a preference for an adjective reading. The mirror tag, AV0-AJ0, again shows adjective-adverb ambiguity, but this time the more likely reading is the adverb.

Table 5.1: BNC Self-Coverage Analysis

Number of Types	Number of Tokens	Percentage of Coverage
500	60500391	61.97
1000	67453571	69.09
2000	74755872	76.57
3000	78936262	80.86
5000	83668773	85.70
8000	87328464	89.45
10000	88817865	90.98
20000	92492214	94.74
30000	94009501	96.29
40000	94842120	97.15
50000	95364884	97.68
60000	95722369	98.05
70000	95981005	98.31
80000	96176051	98.52
90000	96329276	98.67
100000	96452402	98.79
150000	96830203	99.18
200000	97029381	99.39
250000	97155898	99.52
300000	97255898	99.62
350000	97317523	99.68
400000	97367523	99.74
450000	97417523	99.79
500000	97467523	99.84
550000	97517523	99.89
600000	97567523	99.95
652423	97619946	100.00

Some of the words that are assigned ambiguous tags are:

by|| 504969|| AVP|| 371|| AVP-PRP|| 2654|| PRP|| 497746||PRP-AVP||4193|| UNC|| 5

back|| 97154|| AJ0|| 1655|| AJ0-NN1|| 116|| AVP|| 75233|| NN1|| 16910|| NN1-AJ0|| 2102|| NN1-VVB|| 71|| NP0|| 18|| UNC|| 6|| VVB|| 160|| VVB-NN1|| 124|| VVI|| 759

Before going on to further levels of processing, we have tentatively disambiguated the tags by taking first tag in the ambiguous tag as POS tag of that word i.e. in PRP-AVP, PRP is considered as the tag of the word. Although this may cause some problems in frequency estimation, it is inefficient to keep ambiguous tags in the lexicon.

5.1.1.2 Other Preprocessing Steps

We have excluded the types that contain special symbols other than hyphen or space, cardinal numbers, proper names (with only NP0, UNC or both NP0 & UNC tags) from the dictionary. Proper nouns have been removed from the dictionary because they are not finite in number and more specific to particular domains. The words with POS tag UNC have also been removed from the dictionary. After this step, there are 139,204 words in the dictionary which have a coverage of 94.76% on the total BNC corpus.

5.1.1.3 Manual Correction

Closed class words such as pronouns, prepositions, particles, conjunctions, interjections, determiners, cardinal and ordinal numbers have been manually checked. This is very crucial because they are very frequent in language. After this step, the dictionary has 138,401 words.

In BNC corpus, there are only 16 words that are considered as adverb particles. We have extended this particles list after studying the phrasal verbs from the Collins Cobuild dictionary of phrasal verbs.

5.1.1.4 Dictionary Structure

Each line contains a word, its frequency of occurrence in the BNC corpus, all the tags that are assigned in the BNC corpus for the given word, and number of times each particular tag is assigned to the given word in the BNC corpus. Frequency information will be used for developing HMMs. An example entry is shown below. Here we have used || as delimiter to separate fields.

about|| 190615|| PRN|| 7358|| AV0|| 26074|| AVP|| 13037|| PRP|| 139800|| SW2|| 4345

5.1.2 Coverage Analysis on Various Corpora

Dictionary lookup task in syntax will become more cumbersome if the dictionary includes large number of words. Hence it is needless to say, one has to be careful in the selection of words so that dictionary has to be compact at the same time it has to have good coverage.

We have done several experiments to check the coverage of our dictionary. In the first step, We have selected random samples from the BNC corpus. From each file of the BNC corpus, we selected some sentences randomly. Then we have extracted words from the randomly selected corpus. We have performed coverage analysis of the dictionary on this word list. The dictionary has covered 94.83% of the words from the random samples on an average.

We have done analysis of uncovered words in the random samples. We have observed that 37.25% of uncovered words are proper nouns, 29.6% of uncovered words are cardinal numbers, 33.1% are words containing special symbols other than hyphen and space, 0.05% are infrequent words i.e. they occurred less than two times in the entire corpus.

In the second experiment, we have performed coverage analysis of the dictionary on Susanne corpus. Susanne is a manually parsed corpus which is a freely available corpus developed at Oxford University. It contains a subset of Brown Corpus, more precisely 64 files with about 130000 words from 4 categories: Press reportage; Belles letters, biography, memories; scientific and technical writing; adventure and Western fiction. There are nearly 14,200 types and 130,054 tokens in Susanne corpus [123].

Our dictionary has covered 125,328 tokens i.e. 96.37% of the entire Susanne corpus. We have done analysis of uncovered words in Susanne corpus. We have observed that 45.25% of uncovered words are proper nouns, 0.001% of uncovered words are cardinal numbers, 41.2% are words containing special symbols other than hyphen and space, 13% are infrequent words i.e. they occurred less than two times in the entire corpus.

In the third experiment, we have performed coverage analysis of the dictionary words on the Reuters corpus. Reuters corpus[120] contains large quantities of Reuters News stories on different domains. This corpus is made up of 984 MB of newspaper articles in compressed format from issues of Reuters between the 20th Aug., 1996 and 19th Aug., 1997. The number of total news articles is 806,791, which contain 9,822,391 paragraphs, 11,522,874 sentences. There are nearly 15,63,077 types and 18,37,41,416 tokens in Reuters corpus [122].

The dictionary extracted from BNC corpus has covered 15,79,28,745 tokens i.e. 86% of entire Reuters corpus. We have done analysis of uncovered words in Reuters corpus. We have observed that 53.2% of uncovered words are cardinal numbers, 17.2% are words containing special symbols and abbreviations, 29.6% are proper nouns and other words.

In the fourth experiment, we have performed coverage analysis of the dictionary words on word lists collected from different dictionaries. The list is collected from various sources such as dictionary of the Link parser, Collins parser dictionary, Ispell word list and UK word list downloaded from the web. There are totally 95,439 words in this list. The dictionary has covered 73.7% words of the list. We have done some analysis of uncovered words. We have observed that 4.2% of uncovered words are possessives, 5.9% are cardinal numbers, 3.9% are hyphenated words, 8% are pronouns and misspelled words. We have found that 2 to 3% of the valid words are not covered.

We conclude that our dictionary is very good in terms of coverage over a wide variety of usage.

5.2 Sentence Boundary Disambiguation system for English

In this section, we describe our experiments on the sentence boundary disambiguation task.

Sentence boundary disambiguation is a basic task in any kind of Natural Language Processing application. The primary goal of sentence boundary disambiguation is the recognition of sentences, because most linguistic analyzers consider sentences as their units of treatment. Hence it is important for a system to identify the different sentences in a text before performing any language processing application. Many applications rely on texts being segmented into sentences: syntactic parsing, machine translation, text alignment, document summarization. Detecting sentence boundaries is not as trivial as it may appear.

5.2.1 Features

Machine Learning algorithms rely on the extraction and selection of features that adequately characterize the patterns of interest. The task of identifying the features that perform well in a classification algorithm is a difficult one, and the optimal choice can be non-intuitive.

The following were the possible features that we have identified for English sentence segmentation:

- Delimiter: We considered period, exclamation mark, question mark, colon and semicolon as the potential sentence delimiters in English language.
- Prefix: We considered 'prefix' as feature because if a prefix of a sentence boundary delimiter is an abbreviation or proper name, it may or may not indicate a sentence boundary.
- Suffix: We considered 'suffix' as feature because if a suffix of a sentence boundary delimiter is a digit, it may or may not indicate a sentence boundary.
- After word: We considered 'after word' as feature because if the word that comes after a sentence boundary delimiter is part of an abbreviation or starting with a lower case letter, it may or may not indicate a sentence boundary.

We have also formulated some rules to identify abbreviations in the text. For example, a single letter word other than 'i' ending with a period is taken as an abbreviation (Example: P. V. Narasimha Rao). Words without vowels (including 'y') ending with a period are taken as abbreviations.

5.2.2 Training

We have used the British National Corpus (BNC)[16] to identify feature instances for sentence segmentation. BNC Corpus is a 100 million word collection of samples of written and spoken language from a wide range of sources. BNC is sentence tagged and contains total number of s-units just over 6 million (6,053,093). Since BNC is fairly large corpus which is designed to represent a wide range of modern British English of different domains and also sentence tagged, we have decided to use BNC for identifying potential feature instances for the sentence segmentation task.

At each potential delimiter, we have observed the instances of our features that can potentially indicate whether it is sentence boundary or not. We have collected such instances at each delimiter we have considered. We have used the list these instances as training data for our model. We have used an available tool called "Weka" for our training and testing purposes.

Weka[150] is a tool having a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a data set or called from our own Java code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. The main features of Weka are: 1) Comprehensive set of data pre-processing tools, learning algorithms and evaluation methods 2) Graphical user interfaces (including data visualization) 3) Environment for comparing learning algorithms.

We have employed ID3 decision tree and C4.5 Decision tree algorithms for our purpose. The Weka classifier package has its own version of C4.5 known as J48. Since our data is categorical data, decision trees are very much suitable for this task. A decision tree takes as input an object or situation described by a set of properties, and outputs a classification decision.

ID3 is a simple decision tree learning algorithm developed by Ross Quinlan in 1983[141]. The basic idea of ID3 algorithm is to construct the decision tree by employing a top-down, greedy search through the given sets to test each attribute at every tree node. Information gain is used as the metric in order to select the attribute that is most useful for classifying a given sets.

C4.5[116] is an extension of the basic ID3 algorithm designed by Quinlan to address the following issues not dealt with by ID3: 1) Avoiding over fitting the data 2) Determining how deeply to grow a decision tree. 3) Reduced error pruning 4) Rule post-pruning 5) Handling continuous attributes 6) Choosing an appropriate attribute selection measure 7) Handling training data with missing attribute values 8) Handling attributes with differing costs 9) Improving computational efficiency.

5.2.3 Testing

We found that Weka is unstable and crashes for large input data under constraints of memory. We have therefore divided instances of features collected from BNC corpus into 20 different random sets where each set contains 50000 samples. Each test sample has been subjected to 10 fold cross validation. We have observed the F-measure for each random set. On 10 test samples we used ID3 decision tree and on the remaining 10 samples we have used J48 decision tree method. The results are given in table 5.2.

Table 5.2: F-measure of the 20 test samples

method	sample	F-measure (%)	method	sample	F-measure (%)
ID3	sample1	99.0	J48	sample11	98.0
	sample2	99.0		sample12	98.7
	sample3	99.2		sample13	98.8
	sample4	99.1		sample14	98.9
	sample5	99.3		sample15	99.1
	sample6	98.5		sample16	98.3
	sample7	99.2		sample17	98.0
	sample8	99.2		sample18	98.8
	sample9	98.6		sample19	98.9
	sample10	98.5		sample20	99.0
	average	98.9		average	98.65

It may be observed that an overall performance of about 99% has been obtained. This is comparable to the best published results [88] .

5.3 Tag Set

We have studied various tag sets including BNC C5, BNC C7, Susanne and Penn Tree Bank tag sets. The number of tags used are shown in table 5.3.

Table 5.3: Number of POS Tags used in Different Systems

Tag set Name	Number of Tags
BNC C5 tag set	61
BNC C7 tag set	146
BNC C8 tag set	171
Penn TreeBaank tag set	45
Brown tag set	87
SUSANNE tag set	353
Lancaster Oslo/Bergen Corpus	135
London-Lund Corpus	197

Since our work is based on BNC 96 edition with C5 tag set, we have made some extensions as and when required. We have totally 71 tags in the extended tag set. The tag set used in this work is given in the *appendix A*. Here we have given only description about the need for extending the tag set and the news tags that are included into the tag set.

Extensions:

There is no distinction between nominative, accusative and possessive pronouns in the C5 tag set. This distinction is very much required in eliminating many ungrammatical sentences. We have introduced four tags for accusative pronoun(PNA), nominative pronoun(PNN), both nominative and accusative pronoun(PNC) and possessive pronouns(PPS).

There is no distinction distinction between interrogative pronoun and relative conjunction in C5 tag set. Hence, we have added one more tag "CJR" for relative conjunctions.

Pre-determiners occur prior to other determiners. This class of words includes words such as both, half, all, such, what, quite and many. We have introduced new tag for these word called “DTP”.

Some of the adverbs such as ago, each, only can be included into noun groups as post nominal modifiers. Hence we have tagged these kind of words with the tag “NAV”.

Possessive nouns such as Rama’s, Peter’s are tagged with the tag “NPS”.

Special verbs such as ‘need’ and ‘dare’ are tagged as “VS2”, ‘used’ and ‘ought’ are tagged as “VS1” and ‘keep’, ‘go’ and ‘went’ (ex: keep on doing) are tagged as “VS3”.

The word ‘how’ is tagged “AS1” to recognize question adverbs such as ‘how many, how best’.

The word ‘as’ is tagged “AS2” to recognize question adverbs and conjunctions such as ‘as many as, as long as, as soon as’ etc.

The word ‘on’ has a special tag “SW1” when it occurs in verb groups such as ‘keep on doing’.

The word ‘about’ has a special tag “SW2” when it occurs in verb groups such as ‘is about to go’.

Some of the prepositions combine with relative conjunctions, to join two clauses. These prepositions are tagged as “PRN”. For example ‘in which’, ‘by whom’, ‘below which’

Some post-nominal adjectives like ‘payable’ as in ‘the bills payable’ are tagged as “AJBLE”.

We have distributed the frequencies for the newly introduced tags by manual observation of some random samples either from our own manually parsed corpus or the BNC corpus itself. For example, the word ‘which’ is tagged only as “DTQ” in BNC corpus. According to UCSG grammar, it can be either of the three tags, namely, relative conjunction, pronoun and determiner. We have taken examples from manual parsed corpus and studied the distribution of tags in manual parsed corpus. We found that 60% of the times the word ‘which’ is tagged as “CJR”, 25% of the times as “PNQ” and 15% of the times as “DTQ”. The frequencies are distributed accordingly.

5.4 Manually Parsed Corpus Development

We have developed a manually parsed corpus of 4000 sentences, covering a wide variety of sentence structures. 1000 sentences have been randomly selected from BNC corpus, 1065 sentences from ‘Guide to Patterns and Usage in English’ (hereinafter referred to as GPUE corpus) [56] and 1935 sentences from CoNLL-2000 test data. This corpus is thus very useful for evaluating the various modules of the parsing architecture and also for bootstrapping.

This corpus was developed by parsing the sentences using the UCSG shallow parser for English and then manually checking the top parse and making corrections where required. We felt this was far easier than parsing the sentences entirely by hand.

5.5 Preprocessing Steps: Tagging

In the preprocessing step, plain sentences are tagged using the dictionary. Here, we have considered all possible tags in the dictionary for a given word. In case, the word is not found in the dictionary we have used morphological rules to find its tag. The most important aspects of inflectional morphology of English including plurals for nouns, past tense, gerundial and participial forms of verbs and degrees of comparison for adjectives are handled. Derived forms are directly found in the dictionary.

The following are the most productive rules for generating inflectional forms in English:

- plural forms of noun and -s form of lexical verbs
- Superlative forms of adjectives (e.g. oldest, hottest, gravest)
- Comparative forms of adjectives (e.g. better, older)

- -ing forms of lexical verbs (e.g. forgetting, living, returning)
- Past and Past participle forms of lexical verbs (e.g. lived, returned, whetted)

Finally, if the word is directly not found in the dictionary and the root of that word from morphological analysis also not found in the dictionary, we have considered the word as proper noun and assigned NP0 tag for the word.

A POS tagger can be included.

5.6 Finite State Grammar

We have developed a nondeterministic finite state grammar for identifying English word groups. The Finite State Machine has a total of 50 states of which 24 are final states. We have given UCSG finite state grammar for verb groups in the form of transition tables B.1 and B.2 in Appendix B.

5.6.1 Example

Sentence: The sun rises in the east.

Actual word groups in the given sentence

```
<ng><0-2><AT0><the>##<NN1><sun>
<vg><2-3><VVZ><rises>
<ng><3-6><PRP><in>##<AT0><the>##<NN1><east>
```

The following word groups are produced by our FSM:

```
<ng><0-2><AT0><the>##<NN1><sun>
<ng><0-3><AT0><the>##<NN1><sun>##<NN2><rises>
<ng><0-6><AT0><the>##<NN1><sun>##<NN2><rises>##<PRP><in>
##<AT0><the>##<NP0><east>
<ng><0-6><AT0><the>##<NN1><sun>##<NN2><rises>##<PRP><in>
##<AT0><the>##<NN1><east>
<ng><1-2><NN1><sun>
```

```
<ng><1-3><NN1><sun>##<NN2><rises>
<ng><1-6><NN1><sun>##<NN2><rises>##<PRP><in>##<AT0><the>
##<NN1><east>
<ng><1-6><NN1><sun>##<NN2><rises>##<PRP><in>##<AT0><the>
##<NP0><east>
<vg><2-3><VVZ><rises>
<ng><2-3><NN2><rises>
<vg><2-4><VVZ><rises>##<AVP><in>
<ng><2-6><NN2><rises>##<PRP><in>##<AT0><the>##<NN1><east>
<ng><2-6><NN2><rises>##<PRP><in>##<AT0><the>##<NP0><east>
<part><3-4><AVP><in>
<ng><3-6><PRP><in>##<AT0><the>##<NN1><east>
<ng><3-6><PRP><in>##<AT0><the>##<NP0><east>
<ng><4-6><AT0><the>##<NN1><east>
<ng><4-6><AT0><the>##<NP0><east>
<ng><5-6><NN1><east>
<ng><5-6><NP0><east>
```

Here, one can observe that FSM has produced correct word groups and also many other possible word groups.

5.6.2 Evaluation

We have evaluated the performance of the FSM module on various corpora - Susanne parsed corpus, CoNLL 2000 test data set and on our manually parsed corpus of 4000 sentences. The evaluation criteria is *Recall* alone since the aim here is only to include the correct chunks.

The Susanne corpus [123] is a manually parsed corpus containing about 130,000 words in 6891 sentences. Some preprocessing was necessary. Ambiguities with apostrophes have been resolved. Spelling errors mentioned in the Susanne documentation have been corrected. Since the structure of the parse output in the Susanne corpus differs somewhat from that of UCSG, suitable mapping schemes had to be developed and validated [99]. Plain text sentences were extracted and given as input to the UCSG shallow parser.

In Susanne Corpus, phrases are classified into eight types [123] namely, verb phrase, noun phrase, adjective phrase, adverb phrase, prepositional phrase, determiner phrase, numeral phrase, genitive phrase.

Results are given in table 5.4 for Noun, Verb, Adjective and Adverb groups.

Table 5.4: Performance of the Finite State Parser on Susanne Corpus

Word Group Type	No. of Groups in Test Data	No. of Groups Recognized	% Recall
Noun Group	47735	41016	85.92
Verb Group	17559	17179	97.83
Adjective Group	2619	1733	66.17
Adverb Group	5516	4701	85.22
Overall	73429	64629	88.02

Overall, 88.02% of phrases in the Susanne corpus have been correctly identified. 97.83% of all the verb groups could be correctly identified. Failures in the case of verb groups are limited to complex cases such as “have never, or not for a long time, had”.

We have done analysis of the word groups that are not covered by our FSM grammar. The main reason for failures we found is that in Susanne corpus the definition of phrases are very much different from the chunks we are using here. Some phrases in Susanne corpus are recursive in nature. We have given a few examples of failures here.

The examples given below are the noun phrases in Susanne corpus, which include other phrases or clauses within the noun phrases themselves.

- $< ng >$ of the little pink woman who chose to be called auntie
- $< ng >$ the largest majority given a candidate in recent years
- $< ng >$ in a society deeply fissured by antagonisms

The examples given below are the adjective phrases in Susanne corpus, which include other phrases within the adjective phrases themselves.

- $< ajg >$ comfortable about her child

- $< ajg >$ as neat as i can

As another example of the kinds of differences, the word "today" is considered as noun in UCSG dictionary, where is it is treated as adverb in Susanne corpus.

The CoNLL 2000 test data set consists section 20 of the Wall Street Journal corpus (WSJ) and includes 47377 words and 23852 chunks. In the current evaluation, LST chunks (list items) have been excluded. Also, in the UCSG framework, there are no separate PPs - PPs are included in noun groups. Table 5.5 gives the performance in the first set of experiments [75].

Table 5.5: Evaluation of Finite State Parser on CoNLL 2000 Test Data Set

CoNLL Chunk Type	UCSG Terms	Chunks in Test Data	Chunks Recognized	% Recall % Recall
NP	ng	12422	10588	85.24
VP	vg,infg,vgs	4658	3786	81.28
ADVP	avg	866	698	80.60
ADJP	ajg,ags	438	398	90.87
SBAR	sub,rel	535	507	94.77
PRT	part	106	105	99.06
CONJP	sub	9	9	100.00
INTJ	intg	2	1	50.00
Total		19036	16092	84.53

There are a few minor differences in the way chunks are defined in the CoNLL 2000 chunking task and UCSG. Punctuation marks are removed by a pre-processor and handled separately elsewhere in UCSG. Currency symbols such as $ and # are considered part of numbers in UCSG while they become separate words in CoNLL. CoNLL splits chunks across the apostrophes in genitives as in *Rockwell International Corporation's tulsa unit* while UCSG does not. To-infinitives as in *continue to plummet* are recognized separately in UCSG while they may form part of a VP in CoNLL. Also, in keeping the UCSG philosophy, PPs are not recognized separately in UCSG, they are included in noun groups. In order to get a better feel for the true performance of the UCSG shallow parser, the above differences were discounted for and performance is checked again. The results are given in Table 5.6. There is no change in the performance for other groups. Overall, 18185 out of 19130 chunks have been correctly identified, giving a Recall of 95.06%.

Table 5.6: Evaluation of the Finite State Parser on CoNLL Data Set after mapping

CoNLL Chunk Type	UCSG Terms	Chunks in Test Data	Chunks Recognized	Recall (%)
NP,PP	ng	12261	11605	94.65
VP	vg	4283	4223	98.60
-	infg	625	610	97.6
ADVP	avg	866	710	82.56
ADJP	ajg	438	414	94.52
SBAR	sub	544	517	95.03
PRT	part	106	105	99.06
INTJ	intg	2	1	50

Table 5.7 gives the performance of the FSM module on the manually parsed corpus. From the table 5.7, we can observe that very high recall (99.56%) is achieved on manually parsed corpus.

Table 5.7: FSM Evaluation on Manually Parsed Corpus

Chunk type	Symbol	No of Chunks in Corpus	No. of Chunks Found	Recall (%)
Noun	ng	15648	15627	99.86
Verb	vg	6827	6817	99.85
Adverb	avg	908	836	92.07
Adjective	ajg	869	863	99.31
Coordinate conjunction	coord	460	457	99.35
Subordinate conjunction	sub	1048	1048	100
Relative conjunction	rel	460	460	100
Particle	part	31	31	100
To infinitive	infg	955	948	99.27
Interjection	intg	7	7	100
Adjective special	ags	15	15	100
Verb special	vgs	475	475	100
Total	-	27703	27584	99.56

We have done analysis of the word groups that are not covered by our FSM grammar. The main reason we found that in CoNLL corpus, some of the words have tag differences. For example, the word “according to” is a single preposition in UCSG dictionary where as the words are tagged as separate prepositions in CoNLL corpus. Multi-token adverbs such as ‘at last’, ‘no longer’ are not identified by our grammar as on date. There are also tag differences between CoNLL and UCSG tag set. We have considered the word ‘today’ as noun in our dictionary whereas in CoNLL it is considered as adverb.

The table 5.8 shows the number of extra phrases produced by the over generalization of FSM grammar. In manually parsed corpus, there are 27703 correct chunks.

Table 5.8: Analysis of FSM Module - Test Data of 4000 sentences having 27703 phrases in Manually Parsed Corpus

	Plain	POS tagged
Number of phrases produced by FSM module	313306	136926
% of correct chunks recognized by FSM module	99.56	99.96

We may conclude that our finite state grammar is very good in recognizing the correct chunks in most cases. By design, the FSM also produces other possibilities and the UCSG architectures provides a separate module for rating and ranking the chunks produced by the FSM so that the best ones can be selected for further processing.

5.7 Developing HMMs

HMMs were initially developed from 3.7 Million POS-tagged sentences taken from the BNC corpus. Sentences with more than 40 words were excluded. Since we use an extended C5 tag set, POS tags had to be mapped to the extended set where necessary. HMM parameters are estimated from the chunks produced by the Finite State grammar, taking all chunks to be equi-probable. Separate HMMs are built for noun groups, verb groups, adjective groups, adverb groups, infinitive groups and one HMM for all other chunk types.

For example, noun group HMM is estimated using all the noun groups that are produced by parsing 3.7 Million POS tagged sentences using finite state grammar. Here, we have not only the appropriate word groups but also many other possible word groups produced due to lexical ambiguities. Hence, we have considered all possible word groups given by finite state grammar as equiprobable and we have estimated the HMM parameters by taking the ratios of frequency counts. Since, we know the type of the word group from the FSM output, we have estimated Π matrix by observing the frequencies of tags that are able to start noun groups in the entire corpus. The A matrix is estimated from the frequencies of tag to tag transitions within the word groups.

Example Word Group

```
<ng><0-6><AT0><the>##<NN1><sun>##<NN2><rises>##<PRP><in>
##<AT0><the>##<NP0><east>
```

In the above word group, the transition from AT0 tag to NN1 tag has occurred one time. The transition from NN1 tag to NN2 tag has occurred one time. In this way, we have estimated transition frequencies from one tag to the other tag in the A matrix. B matrix contains frequency of observing a particular word or emission symbol at particular state i.e. POS tag. The B matrix values are estimated from a dictionary that includes frequency counts for each word in every possible category. See example dictionary entries in section 5.1.

The probability of a given chunk $P(O,Q|\lambda)$ has been calculated using the equation

$$P(O,Q|\lambda) = \pi_{q_1} b_{q_1}(o_1) a_{q_1,q_2} b_{q_2}(o_2) a_{q_2,q_3} \cdots a_{q_{t-1},q_t} b_{q_t}(o_t) \tag{5.1}$$

where q_1 ,q_2, $\cdots$, q_t is a state sequence, o_1 , o_2,$\cdots$, o_t is an observation sequence.

The chunks ranked accordingly. The mean rank scores and recall are given in table 5.9.

Table 5.9: Performance of the HMM Module on Various Test Data Sets

Corpus	Plain Sentences		POS tagged Sentences	
	Mean Rank Score	Recall (%)	Mean Rank Score	Recall (%)
GPUE	1.65	99.60	1.24	99.98
BNC	2.17	99.36	1.52	99.93
CoNLL	2.46	99.62	1.68	99.73
Total Corpus	2.26	99.56	1.57	99.84

It can be seen that the Recall is high, indicating that the correct chunks were produced most of the times. The Mean Rank Score is also quite close to 1, indicating that the correct chunks tend to cluster near the top.

It is also interesting to observe the Recall and Mean Rank Score within the top k ranks, where k is a given cutoff rank. Table 5.10 and Table 5.11 show that there is a clear tendency for the correct chunks to bubble up close to the top. For example, more than 95% of the correct chunks were found within the top 5 ranks. Nearly 99% of the correct chunks are within a rank of 10.

Table 5.10: Performance of the HMM Module on the Manually Parsed Corpus of 4000 sentences - Plain Sentences as Input

Cutoff	Mean Rank Score	Cumulative Recall (%)
1	1	43.06
2	1.38	69.50
3	1.67	84.72
4	1.85	91.69
5	1.96	95.13
6	2.04	96.91
7	2.08	97.80
8	2.12	98.39
9	2.14	98.70
10	2.16	98.93

Table 5.11: Performance of the HMM Module on the Manually Parsed Corpus of 4000 sentences - POS tagged Sentences as input

Cutoff	Mean Rank Score	Cumulative Recall (%)
1	1	62.74
2	1.28	86.97
3	1.43	95.64
4	1.50	98.31
5	1.54	99.25
6	1.55	99.61
7	1.56	99.72
8	1.56	99.79
9	1.57	99.81
10	1.57	99.82

In the table 5.12, we have shown top ranked chunks that have been selected from the chunk groups having various sizes. Here, size indicates the number of chunks in that group. From the table 5.12 and figure 5.2, we can see that HMM module is able to push the correct chunks towards the top even though the chunk group size is large.

Table 5.12: Performance of the HMM Module on the Manually Parsed Corpus - Plain Sentences

Chunk Group Size	No. of Chunks Selected		Chunk Group Size	No. of Chunks Selected
1	6394		28	2
2	1957		29	2
3	1710		30	2
4	533		31	1
5	416		32	4
6	209		33	1
7	157		34	2
8	113		36	1
9	97		37	1
10	54		40	2
11	31		42	1
12	44		43	1
13	37		44	1
14	24		47	3
15	24		48	2
16	19		51	1
17	14		52	1
18	8		53	2
19	7		55	1
20	10		57	1
21	7		64	1
22	7		65	1
23	5		69	1
24	5		91	1
25	3		100	1
26	4		109	1
27	1		110	1
			233	1

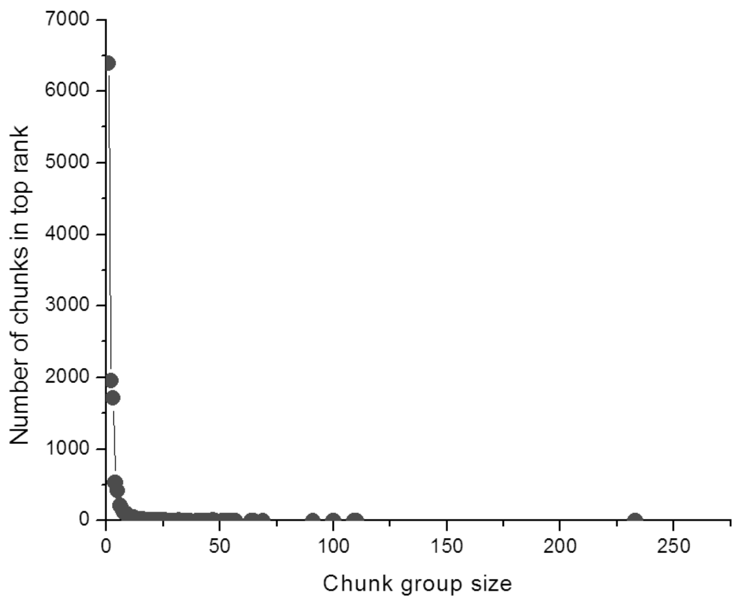


Figure 5.2: Analysis of number of chunks selected in top rank from various chunk group sizes

5.7.1 Example

Sentence: The sun rises in the east.

The following are the word groups and their ranks given by HMM module. Here, the probability values are in logarithmic scale. Each entry includes the chunk type, the starting and ending positions, the chunk itself with the POS tags of all the words, log probability given by HMM, rank, number of items in the set, and the serial number of the branching points.

```
<ng><0-2><AT0><the>##<NN1><sun> <-10.8199668891226><1><4><1>

<ng><0-3><AT0><the>##<NN1><sun>##<NN2><rises>
<-22.645126557751><2><4><1>

<ng><0-6><AT0><the>##<NN1><sun>##<NN2><rises>##
<PRP><in>##<AT0><the>##<NN1><east> <-35.0961918977221><3><4><1>

<ng><0-6><AT0><the>##<NN1><sun>##<NN2><rises>##
<PRP><in>##<AT0><the>##<NP0><east> <-36.6074325860112><4><4><1>
```

```
<vg><2-3><VVZ><rises>   <-10.3267169484799><1><5><2>

<ng><2-3><NN2><rises>   <-11.7411565945832><2><5><2>

<vg><2-4><VVZ><rises>##<AVP><in> < -16.744490507491><3><5><2>

<ng><2-6><NN2><rises>##<PRP><in>##<AT0><the>## <NN1><east>
<-24.1922219345543><4><5><2>

<ng><2-6><NN2><rises>##<PRP><in>##<AT0><the>## <NP0><east>
<-25.7034626228434><5><5><2>

<part><3-4><AVP><in>    <-5.39011993798651><1><3><3>

<ng><3-6><PRP><in>##<AT0><the>##<NN1><east>
<-13.3023793305427><2><3><3>

<ng><3-6><PRP><in>##<AT0><the>##<NP0><east>
<-14.8136200188319><3><3><3>

<ng><4-6><AT0><the>##<NN1><east> <-10.6864467975293><1><2><4>

<ng><4-6><AT0><the>##<NP0><east> <-12.1976874858185><2><2><4>
```

It may be noted from the above example that the correct chunks have been ranked at 1, 1 and 2 respectively.

We have also done some experiments to see the effect of the size of training data used to build HMMs on HMM performance. We have found that as we use more and more training data, the HMM performance is improving significantly. Since we are dealing with very large data sets, even a change in the second decimal place is very significant. The results are shown in table 5.13.

Table 5.13: Effect of the size of training data on HMM performance

Size of the data (No. of sentences)	Mean Rank
0.1Million	2.29
1 Million	2.27
3.7 Million	2.26

5.8 Parse Generation and Ranking

The parse generation module has been evaluated on the manually parsed corpus in terms of rank of the fully correct parse and also in terms of percentage of correct chunks in the top parse. Plain sentences and POS tagged sentences have been considered separately for input. The results are summarized in table 5.14. Here, we have restricted the parsing time taken by the best first search algorithm to 3 epoch seconds for each sentence because the time and space complexity increases exponentially as branching factor (b) and length of the sentence (n) increases. From the tables, we can see that when we restrict best first search module to give best five parses and time limit to 3 epoch seconds, we have 45.52% correct parses within top 5 for plain sentences and 68.02% of correct parses within top 5 for POS tagged sentences. The total number of sentences parsed by the best first search module is only 54.67% for plain sentences and 86.45% for the POS tagged sentences within the stipulated time. It must be noted that since the finite state grammar is recognizing correct chunks with a very high recall and since the HMM modules are used only for ranking and no pruning is done, correct parses will surely be generated in most cases provided we have no time limits.

Table 5.14: Performance of the Best First Search Module - Test Data of 4000 Sentences

Rank	No. of correct Parses	
	(Plain Sentences)	(POS tagged Sentences)
1	1130	1774
2	351	487
3	185	194
4	85	137
5	70	129
% of Correct parses in top 5	45.52	68.02
% of Correct chunks in top parse	78.70	78.42
% of sentences parsed by BFS	54.67	86.45
Time taken to parse	1h:55m:33sec	0h:31m:49sec

We have analyzed the complexity involved in exhaustive search to produce all the parses for a given sentence. We have summarized the results in tables 5.15 and 5.16. We can see that the total number of parses for each sentence increases exponentially with the length of the sentence and also branching factor. The results have also shown that POS tagging greatly helps in parsing by reducing the complexity.

Table 5.15: Analysis of Complexity - Plain Sentences

corpus	Average Sentence Length	Average No. of Parses
GPUE	7.02	381
BNC	15.78	12,428,029
CoNLL	20.06	786,473,522,192

Table 5.16: Analysis of Complexity - POS Tagged Sentences

corpus	Average Sentence Length	Average No. of Parses
GPUE	7.02	5
BNC	15.78	401
CoNLL	20.06	81,794

It may be noted that the performance of the parser in terms of its ability to produce the correct parse is limited only by the Finite State Grammar and the dictionary, since the other modules do not resort to any pruning. However, it is conceivable that in practical usage, we may impose a cutoff and attempt to produce only the top k parses. In this latter case, the percentage of cases where the fully correct parse is included could also be observed.

5.8.1 Example

Sentence: The sun rises in the east.

The following parses are the ranked order given by BFS module if we use dictionary tags

```
<ng>[<AT0><the>##<NN1><sun>##<NN2><rises>##<PRP><in>##<AT0><the>##
<NN1><east>]</ng>   -- -35.2345922674581

<ng>[<AT0><the>##<NN1><sun>]</ng> <vg>[<VVZ><rises>]</vg>
<ng>[<PRP><in>##<AT0><the>##<NN1><east>]</ng>       --
-35.9226802717702

<ng>[<AT0><the>##<NN1><sun>]</ng>
<ng>[<NN2><rises>##<PRP><in>##<AT0><the>##<NN1><east>]</ng>       --
-36.504440120609

<ng>[<AT0><the>##<NN1><sun>##<NN2><rises>##<PRP><in>##<AT0><the>##
<NP0><east>]</ng>   -- -36.7458329557472

<ng>[<AT0><the>##<NN1><sun>]</ng> <vg>[<VVZ><rises>]</vg>
```

```
<ng>[<PRP><in>##<AT0><the>##<NP0><east>]</ng>       --
-37.4339209600594

<ng>[<AT0><the>##<NN1><sun>##<NN2><rises>]</ng>
<ng>[<PRP><in>##<AT0><the>##<NN1><east>]</ng>       --
-37.4397571852259

<ng>[<AT0><the>##<NN1><sun>]</ng>
<ng>[<NN2><rises>##<PRP><in>##<AT0><the>##<NP0><east>]</ng>       --
-38.0156808088982

<ng>[<AT0><the>##<NN1><sun>]</ng> <ng>[<NN2><rises>]</ng>
<ng>[<PRP><in>##<AT0><the>##<NN1><east>]</ng>       --
-38.7096050383768

<ng>[<AT0><the>##<NN1><sun>##<NN2><rises>]</ng>
<ng>[<PRP><in>##<AT0><the>##<NP0><east>]</ng>       --
-38.950997873515

<ng>[<AT0><the>##<NN1><sun>]</ng> <vg>[<VVZ><rises>##<AVP><in>]</vg>
<ng>[<AT0><the>##<NN1><east>]</ng>       -- -39.724521297768

<ng>[<AT0><the>##<NN1><sun>]</ng> <ng>[<NN2><rises>]</ng>
<ng>[<PRP><in>##<AT0><the>##<NP0><east>]</ng>       --
-40.220845726666

<ng>[<AT0><the>##<NN1><sun>]</ng> <vg>[<VVZ><rises>##<AVP><in>]</vg>
<ng>[<AT0><the>##<NP0><east>]</ng>       -- -41.2357619860571

<ng>[<AT0><the>##<NN1><sun>##<NN2><rises>]</ng>
<part>[<AVP><in>]</part> <ng>[<AT0><the>##<NN1><east>]</ng>       --
-46.222079476789

<ng>[<AT0><the>##<NN1><sun>]</ng> <ng>[<NN2><rises>]</ng>
<part>[<AVP><in>]</part> <ng>[<AT0><the>##<NN1><east>]</ng>       --
-47.4919273299399
```

```
<ng>[<AT0><the>##<NN1><sun>##<NN2><rises>]</ng>
<part>[<AVP><in>]</part> <ng>[<AT0><the>##<NP0><east>]</ng>      --
-47.7333201650782

<ng>[<AT0><the>##<NN1><sun>]</ng> <ng>[<NN2><rises>]</ng>
<part>[<AVP><in>]</part> <ng>[<AT0><the>##<NP0><east>]</ng>      --
-49.0031680182291

<ng>[<AT0><the>##<NN1><sun>]</ng> <vg>[<VVZ><rises>]</vg>
<part>[<AVP><in>]</part> <ng>[<AT0><the>##<NN1><east>]</ng>      --
-62.3971218959318

<ng>[<AT0><the>##<NN1><sun>]</ng> <vg>[<VVZ><rises>]</vg>
<part>[<AVP><in>]</part> <ng>[<AT0><the>##<NP0><east>]</ng>      --
-63.908362584221
```

It may be observed from the above example that there are 18 parses and the fully correct parse is in rank two.

We have also implemented a modified beam search algorithm to improve the parser efficiency in terms of time and space. Here, we have kept a threshold on the probability so that the word groups which are having probability less than the threshold can be pruned. Please note that here the HMM probabilities are in logarithmic scale. In this way, we can reduce the number of combinations the parser has to explore and also save a good deal of memory. But this may cause pruning of some of correct parses. If we do not want to loose the correct parse, we have to increase the threshold accordingly. As the threshold increases, the complexity approaches that of the best first search. The results in table 5.17 have been obtained for a beam threshold of 1.

Table 5.17: Performance of the modified Beam Search - Test Data of 4000 Sentences

Rank	No. of correct Parses	
	Plain	POS tagged
1	1262	1796
2	259	240
3	67	36
4	35	22
5	9	4
% of Correct parses in top 5	40.8	52.45
% of Correct chunks in top parse	67.98	74.31
% of sentences parsed by modified Beam Search	100.00	99.87
Time taken to parse	0h:15m:31sec	0h:0m:18sec

We have also studied effect of increase in threshold on parse generation. As we increase threshold, the performance approaches that of the best first search, but the time taken to parse will also increase. The results are shown in table 5.18.

Table 5.18: Performance of the modified Beam Search with increasing threshold - Test Data of 4000 Sentences

	Threshold 1	Threshold 3
1	1796	1796
2	240	497
3	36	200
4	22	142
5	4	119
% of Correct parses in top 5	52.45	68.85
% of Correct chunks in top parse	74.31	74.31
Time taken to parse	0h:0m:18sec	6h:07m:20sec

We have also studied the percentage correct tags assigned to the words in the top parse of modified beam search module. We have observed that 96.01% of the words are assigned correct POS tags in the top parse. This shows that most of the times the top parse given by the parse generation module is almost correct in terms of POS tags and may only have problems with chunk boundary detection. The results are shown in table 5.19.

Table 5.19: Evaluation of the POS tags in the top parse of parse generation module (modified beam search)

Number of words	62268
Number of words assigned Correct POS tags:	59784
% of correct POS tags	96.01

It may be observed that the only kind of linguistic constraints we have used so far is the structure of chunks as captured by the Finite State Grammar. It is in fact interesting to see fully correct parse (that is, chunk sequence) being produced by the system in many cases before applying any sentence level linguistic constraints at all. We have not included a grammar of clause structure, hierarchical structure of clauses and phrases in sentences, or functional structure constraints such as sub-categorization and selectional restrictions or even simple agreement rules. Further improvements to the parser performance will critically depend on judicious application of relevant linguistic constraints within the overall architecture.

Also, more work is needed to assign thematic roles to the chunk sequences produced by the parser.

5.9 Bootstrapping

We hypothesize that the HMM parameters can be refined through bootstrapping. Initial HMMs were developed from chunks produced by the over-general Finite State Grammar, taking all chunks to be equi-probable. Once the HMMs have been built, we can use the same HMMs to rate and rank the chunks and further produce parses using best first search. From the results obtained, it is clear that the top ranked chunks and chunks from the top ranked parses will give us better data for re-building HMMs. The new data sets so generated contain a higher percentage of correct chunks. In other words, noise is reduced. However, the size of the data set also comes down as shown in table 5.20.

Table 5.20: Bootstrapping: Data Set Size

HMM development Phase	No. of Sentences	No. of Chunks
Initial HMM building with FSM Output	3770917	122748054
Bootstrapping with HMM Top Ranked Chunks	2008877	22368823
Bootstrapping with Best first search Top Parse	1804827	11061598

To prove the bootstrapping hypothesis, we have carried out several experiments. Plain text sentences from BNC corpus, 5 to 20 words in length, have been used. All possible chunks are obtained using the Finite State State Grammar-Parser and HMMs built from these chunks. In one experiment, only the chunks rated highest by these very HMMs are taken as training data for bootstrapping. In a second experiment, best first search is also carried out and chunks from the top ranked parse alone are taken for bootstrapping. In a third experiment, data from these two sources have been combined. Best results were obtained when the chunks from the top parse alone were used for bootstrapping. Tables 5.21 and 5.22 shows the effect of bootstrapping on HMM module.

Table 5.21: Effect of Bootstrapping after iteration-1: on 4000 sentences from Manually Parsed Corpus containing a total of 27703 chunks

Cutoff	Initial			Iteration-1		
	No. of Chunks	Recall	Mean Rank	No. of Chunks	Recall	Mean Rank
1	11929	43.06	1.0	12611	45.52	1.0
2	19254	69.50	1.38	19787	71.43	1.36
3	23470	84.72	1.67	23609	85.22	1.63
4	25402	91.69	1.85	25418	91.75	1.80
5	26356	95.13	1.96	26303	94.94	1.90
6	26848	96.91	2.04	26805	96.75	1.98
7	27096	97.80	2.08	27078	97.74	2.03
8	27257	98.39	2.12	27226	98.28	2.06
9	27344	98.70	2.14	27326	98.63	2.09
10	27406	98.93	2.16	27393	98.88	2.11

Table 5.22: Effect of Bootstrapping after iteration-2: on 4000 sentences from Manually Parsed Corpus containing a total of 27703 chunks

Cutoff	Iteration-1			Iteration-2		
	No. of Chunks	Recall	Mean Rank	No. of Chunks	Recall	Mean Rank
1	12611	45.52	1.0	13090	47.25	1
2	19787	71.43	1.36	20170	72.81	1.35
3	23609	85.22	1.63	23811	85.95	1.60
4	25418	91.75	1.80	25541	92.20	1.77
5	26303	94.94	1.90	26401	95.30	1.87
6	26805	96.75	1.98	26863	96.97	1.94
7	27078	97.74	2.03	27108	97.85	1.99
8	27226	98.28	2.06	27249	98.36	2.02
9	27326	98.63	2.09	27336	98.68	2.04
10	27393	98.88	2.11	27407	98.93	2.06

It may be observed that the percentage of correct chunks is increasing in the top 4 positions and decreasing thereafter, clearly showing that bootstrapping has helped to rate and rank chunks better.

There is also some improvement in the final parse when the HMMs obtained through bootstrapping are used. See tables 5.23 and 5.24.

Table 5.23: Effect of Bootstrapping on Parse Generation - Plain Sentences (Best First Search - Epoch Time limit 3)

Rank	No. of correct Parses		
	Initial	Iteration-1	Iteration-2
1	1130	1172	1210
2	351	308	352
3	185	152	157
4	85	82	83
5	70	72	68
% of Correct parses in top 5	45.52	44.65	46.75
% of Correct chunks in top parse	78.70	83.17	83.92

Table 5.24: Effect of Bootstrapping on Parse Generation - POS Tagged Sentences (Best First Search - Epoch Time limit 3)

Rank	No. of correct Parses		
	Initial	Iteration-1	Iteration-2
1	1774	2113	2193
2	487	470	495
3	194	186	164
4	137	132	129
5	129	89	91
% of Correct parses in top 5	68.02	74.75	76.80
% of Correct chunks in top parse	78.42	87.51	88.26

5.9.1 Example

Sentence: The sun rises in the east.

The following are the top 5 parses in ranked order given by BFS module after bootstrapping if we use dictionary tags

```
<ng>[<AT0><the>##<NN1><sun>]</ng> <vg>[<VVZ><rises>]</vg>
<ng>[<PRP><in>##<AT0><the>##<NP0><east>]</ng>      --
-37.1886283215909

<ng>[<AT0><the>##<NN1><sun>]</ng> <vg>[<VVZ><rises>]</vg>
<ng>[<PRP><in>##<AT0><the>##<NN1><east>]</ng>      --
-38.8822563306516

<ng>[<AT0><the>##<NN1><sun>##<NN2><rises>]</ng>
<ng>[<PRP><in>##<AT0><the>##<NP0><east>]</ng>      --
-39.357919583567

<ng>[<AT0><the>##<NN1><sun>##<NN2><rises>]</ng>
<ng>[<PRP><in>##<AT0><the>##<NN1><east>]</ng>      --
-41.0515475926276

<ng>[<AT0><the>##<NN1><sun>]</ng> <ng>[<NN2><rises>]</ng>
<ng>[<PRP><in>##<AT0><the>##<NP0><east>]</ng>      --
-41.7994535067038
```

It may be observed that the correct parse is still in second position but the top parse is far better.

In the tables 5.25 and 5.26, we have shown the effect of bootstrapping on modified beam search algorithm results. Interestingly, bootstrapping also improved the performance of beam search. This is due to the fact that the distribution of probabilities among the phrases has improved with bootstrapping.

Table 5.25: Effect of Bootstrapping on Parse Generation - Plain Sentences (modified Beam Search with threshold 1)

Rank	No. of correct Parses	
	Initial	Iteration-2
1	1262	1386
2	259	259
3	67	57
4	35	33
5	9	14
% of Correct parses in top 5	40.08	43.72
% of Correct chunks in top parse	67.98	76.84

Table 5.26: Effect of Bootstrapping on Parse Generation - POS Tagged Sentences (modified Beam search with threshold 1)

Rank	No. of correct Parses	
	Initial	Iteration-2
1	1796	2267
2	240	188
3	36	17
4	22	13
5	4	6
% of Correct parses in top 5	52.45	62.27
% of Correct chunks in top parse	74.31	86.32

The performance figures given above need to interpreted with care. We have seen that the percentage correct tags assigned to the words in the top parse is over 96%. This shows that most of the times the top parse given by the parse generation module is almost correct in terms of POS tags and may only have minor problems with chunk boundary detection. The very definition of chunks is much more demanding in UCSG - we expect prepositions to be combined with the appropriate noun groups, we expect correct handling of adverb particles which may be ambiguous with a preposition, etc. A quick check by manual observation shows that in most cases the top parse is reasonably good if not 100% perfect. Also, the top parse may be more or less adequate for applications such as IE. More thorough examination of this aspect is planned and all that we wish to say now is that one should not be disheartened by the not-so-high performance figures depicted here.

5.10 Comparison with other Systems

1. Plain Sentence:

By the death of Gokhale, a great statesman and patriot was lost to India.

Tagged Sentence:

```
<PRN_AVP_PRP><by>##<AT0><the>##<NN1><death>##<PRN_PRF_AVP><of>##
<NP0><Gokhale>##<PUN><,>##<AT0><a>##<NP0_AJ0><great>##<NN1><statesman>##
<CJC><and>##<NP0_NN1><patriot>##<VBD><was>##<VVN_VVD><lost>##
<PRN_TO0_PRP_AVP><to>##<NP0><India>##
```

UCSG output:

The chunk types in UCSG shallow parsing system are: 1) ng: noun group, 2) vg: verb group, 3) vgs: verb group special, 4) avg: adverb group, 5) ajg: adjective group, 6) ags: adjective group special, 7) coord: coordinate conjunction, 8) sub: subordinate conjunction, 9) rel: relative conjunction, 10) part: particle group, 11) infg: infinitive group, 12) intg: interjection group.

The top 5 parses from UCSG shallow parser in ranked order are shown below. Top parse is fully correct.

```
<ng>[<PRP><by>##<AT0><the>##<NN1><death>##<PRF><of>##
<NP0><gokhale>]</ng> <ng>[<AT0><a>##<AJ0><great>##<NN1><statesman>##
<CJC><and>##<NN1><patriot>]</ng> <vg>[<VBD><was>##<VVN><lost>]</vg>
<ng>[<PRP><to>##<NP0><india>]</ng>

<ng>[<PRP><by>##<AT0><the>##<NN1><death>##<PRF><of>##
<NP0><gokhale>]</ng> <ng>[<AT0><a>##<AJ0><great>##<NN1><statesman>##
<CJC><and>##<NP0><patriot>]</ng> <vg>[<VBD><was>##<VVN><lost>]</vg>
<ng>[<PRP><to>##<NP0><india>]</ng>

<ng>[<PRP><by>##<AT0><the>##<NN1><death>##<PRF><of>##
<NP0><gokhale>]</ng>
<ng>[<AT0><a>##<AJ0><great>##<NN1><statesman>]</ng>
<coord>[<CJC><and>]</coord> <ng>[<NN1><patriot>]</ng>
<vg>[<VBD><was>##<VVN><lost>]</vg>
<ng>[<PRP><to>##<NP0><india>]</ng>

<ng>[<PRP><by>##<AT0><the>##<NN1><death>]</ng>
<ng>[<PRF><of>##<NP0><gokhale>]</ng>
<ng>[<AT0><a>##<AJ0><great>##<NN1><statesman>##
<CJC><and>##<NN1><patriot>]</ng> <vg>[<VBD><was>##<VVN><lost>]</vg>
<ng>[<PRP><to>##<NP0><india>]</ng>

<ng>[<PRP><by>##<AT0><the>##<NN1><death>##<PRF><of>##
<NP0><gokhale>]</ng> <ng>[<AT0><a>##<AJ0><great>##<NN1><statesman>##
<CJC><and>##<NN1><patriot>]</ng>
<vg>[<VBD><was>##<VVN><lost>##<AVP><to>]</vg>
<ng>[<NP0><india>]</ng>
```

Memory Based Shallow Parser Output:

The parse output shown below is taken from web based demo of memory based shallow parser developed by Walter Daelemans et. al.[32, 86].

```
{PNP [Prep By/IN Prep] [NP the/DT death/NN NP] PNP} {PNP [Prep of/IN
Prep] [NP Gokhale//NNP NP] PNP} ,/, [NP a/DT great/JJ statesman/NN
NP] and/CC [NP patriot//NN NP] [VP was/VBD lost/VBN VP] {PNP [Prep
to/TO Prep] [NP India/NNP NP] PNP} ./.
```

Note that the coordinating conjunction ‘and’ has been left out, ‘by the death’ and ‘of Gokhale’ have become separate chunks.

Cognitive Computation Group Shallow Parser Output:

The parse output shown below is taken from web based demo of shallow parser developed by Cognitive Computation Group [27].

```
[PP By]  [NP the death]  [PP of]  [NP Gokhale]  , [NP a great
statesman and patriot]  [VP was lost]  [PP to]  [NP India]  .
```

Observe that prepositions are treated as independent chunks.

2. Plain Sentence:

I have no ulterior motive in offering you help.

Tagged Sentence:

```
<PNN_CRD><i>##<VHB><have>##<AT0_ITJ_AV0_NN1><no>##
<AJ0><ulterior>##<NN1><motive>##<PRN_PRP_AVP><in>##
<VVG><offering>##<PNC><you>##<VVB_NN1><help>##
```

UCSG output:

The top 5 parses from UCSG shallow parser in ranked order are given below. The correct parse is in position 5.

```
<ng>[<PNN><i>]</ng> <vg>[<VHB><have>]</vg>
<ng>[<AT0><no>##<AJ0><ulterior>##<NN1><motive>##
<PRP><in>##<VVG><offering>]</ng> <ng>[<PNC><you>]</ng>
<vg>[<VVB><help>]</vg>

<ng>[<PNN><i>]</ng> <vg>[<VHB><have>]</vg>
<ng>[<AT0><no>##<AJ0><ulterior>##<NN1><motive>]</ng>
<ng>[<PRP><in>##<VVG><offering>]</ng> <ng>[<PNC><you>]</ng>
```

```
<vg>[<VVB><help>]</vg>

<ng>[<PNN><i>]</ng> <vg>[<VHB><have>]</vg>
<ng>[<AT0><no>##<AJ0><ulterior>##<NN1><motive>##
<PRP><in>##<VVG><offering>]</ng> <ng>[<PNC><you>]</ng>
<ng>[<NN1><help>]</ng>

<ng>[<CRD><i>]</ng> <vg>[<VHB><have>]</vg>
<ng>[<AT0><no>##<AJ0><ulterior>##<NN1><motive>##
<PRP><in>##<VVG><offering>]</ng> <ng>[<PNC><you>]</ng>
<vg>[<VVB><help>]</vg>

<ng>[<PNN><i>]</ng> <vg>[<VHB><have>]</vg>
<ng>[<AT0><no>##<AJ0><ulterior>##<NN1><motive>]</ng>
<ng>[<PRP><in>##<VVG><offering>]</ng> <ng>[<PNC><you>]</ng>
<ng>[<NN1><help>]</ng>
```

Memory Based Shallow Parser Output:

```
[NP I/PRP NP] [VP have/VBP VP] [NP no/DT NP] {PNP [Prep ulterior//IN
Prep] [NP motive/NN NP] PNP} {PNP [Prep in/IN Prep] [NP offering/NN
NP] PNP} [NP you/PRP NP] [VP help/VB VP] ./.
```

Observe that ‘no’ has become a noun group containing a single determiner, ‘ulterior motive’ has become a prepositional phrase, and ‘help’ is treated as a verb.

Cognitive Computation Group Shallow Parser Output:

```
[NP I] [VP have] [NP no ulterior motive] [PP in] [VP offering] [NP
you] [VP help] .
```

Here ’offering’ has become a verb group and prepositions are treated as chunks in their own right.

3. Plain Sentence:

Concern for the environment has always topped our agenda.

Tagged Sentence:

```
<VVB_NN1><concern>##<PRN_PRP_CJS_AVP><for>##<AT0><the>##
<NN1><environment>##<VHZ><has>##<AV0><always>##<VVN_VVD><topped>
##<DPS><our>##<NN1><agenda>##
```

UCSG output:

The top 5 parses from UCSG shallow parser in ranked order are given below. Top parse is fully correct.

```
<ng>[<NN1><concern>##<PRP><for>##<AT0><the>##<NN1><environment>]</ng>
<vg>[<VHZ><has>##<AV0><always>##<VVN><topped>]</vg>
<ng>[<DPS><our>##<NN1><agenda>]</ng>

 <ng>[<NN1><concern>]</ng>
<ng>[<PRP><for>##<AT0><the>##<NN1><environment>]</ng>
<vg>[<VHZ><has>##<AV0><always>##<VVN><topped>]</vg>
<ng>[<DPS><our>##<NN1><agenda>]</ng>


<ng>[<NN1><concern>##<PRP><for>##<AT0><the>##<NN1><environment>]</ng>
<vg>[<VHZ><has>]</vg> <ajg>[<AV0><always>##<VVN><topped>]</ajg>
<ng>[<DPS><our>##<NN1><agenda>]</ng>


<ng>[<NN1><concern>##<PRP><for>##<AT0><the>##<NN1><environment>]</ng>
<vg>[<VHZ><has>##<AV0><always>]</vg> <ajg>[<VVN><topped>]</ajg>
<ng>[<DPS><our>##<NN1><agenda>]</ng>
```

```
<ng>[<NN1><concern>##<PRP><for>##<AT0><the>##<NN1><environment>]</ng>
<vg>[<VHZ><has>]</vg> <avg>[<AV0><always>]</avg>
<vg>[<VVD><topped>]</vg> <ng>[<DPS><our>##<NN1><agenda>]</ng>
```

Memory Based Shallow Parser Output:

```
[NP concern/NN NP] {PNP [Prep for/IN Prep] [NP the/DT environment/NN
NP] PNP} [VP has/VBZ always/RB topped/VBN VP] [NP our/PRP agenda/NN
NP] ./.
```

By now it should be very clear as to why it is very important to combine prepositions with noun groups appropriately to get a clear reading of the given sentence. UCSG output is generally far better than the output of other parsing systems.

Cognitive Computation Group Shallow Parser Output:

```
[NP concern] [PP for] [NP the environment] has [ADVP always] [VP
topped] [NP our agenda] .
```

UCSG requires that every word in the given sentence is included in the final parse. Leaving out words like makes the parse output so much less useable.

4. Plain Sentence:

He is one of the authors who are destined to be immortal.

Tagged Sentence:

```
<PNN><he>##<VBZ><is>##<CRD_PNI><one>##<PRN_PRF_AVP><of>
##<AT0><the>##<NN2><authors>##<NP0_CJR_PNQ><who>##<VBB><are>##
<VVN><destined>##<PRN_TO0_PRP_AVP><to>##<VBB><be>##<AJ0><immortal>##
```

UCSG output:

The top 5 parses from UCSG shallow parser in ranked order are given below. Top parse is fully correct. 'who' is treated as a conjunction introducing a relative clause, thereby facilitating extensions to a full parsing system. Compare with the outputs of other parsers below.

```
<ng>[<PNN><he>]</ng> <vg>[<VBZ><is>]</vg>
<ng>[<CRD><one>##<PRF><of>##<AT0><the>##<NN2><authors>]</ng>
<rel>[<CJR><who>]</rel> <vg>[<VBB><are>##<VVN><destined>]</vg>
<infg>[<TO0><to>##<VBB><be>]</infg> <ajg>[<AJ0><immortal>]</ajg>

<ng>[<PNN><he>]</ng> <vg>[<VBZ><is>]</vg>
<ng>[<PNI><one>##<PRF><of>##<AT0><the>##<NN2><authors>]</ng>
<rel>[<CJR><who>]</rel> <vg>[<VBB><are>##<VVN><destined>]</vg>
<infg>[<TO0><to>##<VBB><be>]</infg> <ajg>[<AJ0><immortal>]</ajg>

<ng>[<PNN><he>]</ng> <vg>[<VBZ><is>]</vg> <ajg>[<CRD><one>]</ajg>
<ng>[<PRF><of>##<AT0><the>##<NN2><authors>]</ng>
<rel>[<CJR><who>]</rel> <vg>[<VBB><are>##<VVN><destined>]</vg>
<infg>[<TO0><to>##<VBB><be>]</infg> <ajg>[<AJ0><immortal>]</ajg>

<ng>[<PNN><he>]</ng> <vg>[<VBZ><is>]</vg> <ng>[<CRD><one>]</ng>
<ng>[<PRF><of>##<AT0><the>##<NN2><authors>]</ng>
<rel>[<CJR><who>]</rel> <vg>[<VBB><are>##<VVN><destined>]</vg>
<infg>[<TO0><to>##<VBB><be>]</infg> <ajg>[<AJ0><immortal>]</ajg>


<ng>[<PNN><he>]</ng> <vg>[<VBZ><is>]</vg>
<ng>[<CRD><one>##<PRF><of>##<AT0><the>##<NN2><authors>]</ng>
<ng>[<PNQ><who>]</ng> <vg>[<VBB><are>##<VVN><destined>]</vg>
<infg>[<TO0><to>##<VBB><be>]</infg> <ajg>[<AJ0><immortal>]</ajg>
```

Memory Based Shallow Parser Output:

```
[NP He/PRP NP] [VP is/VBZ VP] [NP one/CD NP] {PNP [Prep of/IN Prep]
[NP the/DT authors/NNS NP] PNP} [NP who/WP NP] [VP are/VBP
destined/VBN to/TO be/VB VP] [ADJP immortal//JJ ADJP] ./.
```

Cognitive Computation Group Shallow Parser Output:

```
[NP He]  [VP is]  [NP one]  [PP of]  [NP the authors]  [NP who] [VP
are destined to be]  [ADJP immortal]  .
```

Only a few simple examples have been included here. It can observed that UCSG Shallow Parse output is generally superior.

5.11 Closing Remarks

Chunkers are usually evaluated just for the percentage of correct chunks they produce, not for the correctness of the complete parse (chunk sequence) for the whole sentence. We have placed greater demands on ourselves and we expect our parser to produce optimal chunk sequence for the whole sentence. No word can be left out and there can be no overlaps either. Further, we produce all (or top few) combinations and that too in hopefully a best first order. Since we are aiming at chunks that correspond to answers to questions that can be asked of the given sentence, the very nature of the chunking task here is more semantic and hence more demanding. More over, we have used a fairly fine grained tag set with more than 70 tags. The data we have started with, namely the BNC POS tagged corpus, has tagging errors, multiple tags are given in many cases, some words are not tagged, and the tag set had to be extended. Given these factors, the performance we are able to achieve both in terms of percentage of correct chunks in the top parse and rank of the fully correct parse is very encouraging. We are demanding perfect match with manually parsed sentences and in most cases we have observed that the top parse is nearly correct. For example, it could be just a NN1/NP0 error in one of the chunks because of which the top parse may not match with correct parse. This is not a serious problem for many applications. Consider the following example:

Plain Sentence:

There is no rose without thorn.

Tagged Sentence:

```
<EX0_AV0><there>##<VBZ><is>##<AT0_ITJ_AV0_NN1><no>##
<NP0_VVD_NN1><rose>##<PRN_AVP_PRP><without>##
<NP0_NN1><thorn>##
```

UCSG output:

```
<ng>[<EX0><there>]</ng> <vg>[<VBZ><is>]</vg>
<ng>[<AT0><no>##<NP0><rose>]</ng>
<ng>[<PRP><without>##<NP0><thorn>]</ng>

<ng>[<EX0><there>]</ng> <vg>[<VBZ><is>]</vg>
<ng>[<AT0><no>##<NP0><rose>##<PRP><without>##
<NP0><thorn>]</ng>

<ng>[<EX0><there>]</ng> <vg>[<VBZ><is>]</vg>
<ng>[<AT0><no>##<NN1><rose>]</ng>
<ng>[<PRP><without>##<NP0><thorn>]</ng>

<ng>[<EX0><there>]</ng> <vg>[<VBZ><is>]</vg>
<avg>[<AV0><no>]</avg>
<vg>[<VVD><rose>]</vg> <ng>[<PRP><without>##
<NP0><thorn>]</ng>

<avg>[<AV0><there>]</avg> <vg>[<VBZ><is>]</vg>
<ng>[<AT0><no>##<NP0><rose>]</ng>
<ng>[<PRP><without>##<NP0><thorn>]</ng>
```

From the above parses, We can see that the first parse is correct parse if we just ignore NN1/NP0 confusion.

The UCSG parser developed for English is a wide coverage shallow parsing system. The system has been built and tested on very large data sets, covering a wide variety of texts, which is giving confidence that the system will perform well on new, unseen texts. The system is general and not domain specific, but we can adapt and fine tune for any specific domain as and when needed. We are confident that wide coverage and robust shallow parsing systems can be developed using the UCSG architecture for other languages of the world as well.

In this thesis, we have not claimed any work on parsing Indian languages because of the lack of language resources. Important resources such as large scale dictionaries with syntactic information are hardly available. Even though there are claims about morphological analyzers, their performance on large scale corpora is unknown for many Indian languages and in particular for Telugu. Hence, we are working on the development of high quality lexical resources including dictionaries and thesauri, morphological analyzers and stemmers, POS tagging etc. We may soon be able to take up development of wide coverage computational grammars and parsers for Telugu and other Indian languages. We believe that the UCSG shallow parsing architecture we have developed will be effective for Indian languages as well.

There are no large scale POS tagged corpora available for Indian languages. Even large scale plain corpora are available only for few major Indian languages [76]. Indian Languages are lagging behind in terms of technology. Language Engineering on a serious scale has started off only recently in India and we need to speed up. Indian languages are also quite different from English and techniques which work well for English may not always work very well for Indian languages. We have carried out variety of statistical analyses on major Indian language DoE-CIIL corpora of about 3 Million words and on a nearly 40 Million word text corpus of Telugu which is developed at University of Hyderabad, to understand the nature of Indian languages [76].

Language identification is one more important task in Indian context since there are many languages and many scripts and mixing up different languages and scripts in the same document is quite common. We have carried out variety of experiments to distinguish effectively between different Indian languages from small text samples [98].

Chapter 6

Conclusions

6.1 Summary of Results

In this thesis, we have proposed a methodology for building wide coverage shallow parsers by a judicious combination of linguistic and statistical techniques without need for large amounts of training corpus to start with. We present an architecture, called UCSG Shallow Parsing Architecture, which uses a judicious combination of linguistic and statistical approaches for building wide coverage shallow parsing systems. The input to the parsing system is one sentence, either plain or POS tagged. Output is an ordered set of parses. The aim is to produce all possible parses in ranked order hoping to get the best parses to the top. In this work, by parse we mean a sequence of chunks. This can be extended to more detailed syntactic analysis. Chunks are sequences of words. A chunk or a "word group" as we prefer to call it in UCSG, is "a structural unit, a non-overlapping and non-recursive sequence of words, that can as a whole, play a role in some predication".

In the UCSG architecture, a Finite State Grammar is designed to accept *all* valid word groups but not necessarily *only* those word groups that are appropriate in context for a given sentence. From the literature we have seen that simultaneous satisfaction of both the *all and only* requirements has proven very difficult in practice. A separate statistical component, encoded in HMMs (Hidden Markov Model), has been used to rate and rank the word groups so produced. Note that we are recognizing chunks using HMMs. Also, we are not pruning, we are only rating and ranking the word groups already produced. Finally we use a best first search module to produce parse outputs in best first order, without compromising on the ability to produce all possible parses. We also propose a bootstrapping strategy for improving HMM parameters as well as the performance of the parser as a whole. We prove the efficacy of the proposed architecture by developing a wide coverage shallow parser for English.

In this work, we have built a dictionary that includes words, POS tags, and, most importantly, frequency of occurrence for each tag for each word starting from the British National Corpus of English. We have also developed a manually parsed corpus of 4000 sentences as per UCSG syntax by taking sentences from a wide variety of sources. We have shown that finite state machines are sufficient to produce all valid word groups for a given sentence. We have evaluated FSM module in terms of the number of correct chunks it can recognize. We have achieved a high recall of 99.5% on manually parsed corpus, 95.06% on CoNLL test data and 88.02% on Susanne corpus. We have successfully built HMMs by using only POS tagged BNC sentences and used them for rating and ranking word groups. We have evaluated performance of the HMM module in terms of mean rank score i.e. mean of the distribution of ranks of the correct chunks in the parsed corpus. We have obtained a good mean rank scores i.e. 2.26 for plain sentences and 1.57 for POS tagged sentences on a test data of 4000 sentences.

We have used a best first search algorithm to select chunk sequences as a parse of given sentence and these parses are given in best first order. When we restrict best first module to give best five parses and time limit to 3 epoch seconds, we have obtained 45.52% correct parses within top 5 for plain sentences and 68.02% of correct parses within top 5 for POS tagged sentences. The percentage of correct chunks in the top parse is 78.70 for plain sentences and 78.42 for POS tagged sentences. We have also used modified beam search method. We gain a lot in terms of computational efficiency but we may loose some of correct parses if there is too much pruning. Interestingly, the number of correct parses in top position for plain sentences has actually increased from 28.25% to 31.55%. The number of sentences that can be parsed within the stipulated time has also increased from 54.67% to 100%.

We have proved our hypothesis of bootstrapping to improve HMMs parameters as well as the performance of the whole parser. We have done bootstrapping in three ways: by taking HMM top ranked chunks, chunks from top parse given by third module and both combined. We have found that bootstrapping from top parses from best first search module gives best results. We are able to improve the mean rank score to 2.21 from 2.26 in first iteration and to 2.16 in the second iteration. The performance of the parser also improved in terms of pushing the correct parses to the top. Before bootstrapping, there were 28.25% of correct parses in the top position for plain sentences and this is improved to 30.25%. The percentage of correct chunks in the top parse has improved from 78.70% to 83.92%. For tagged sentences as input, the percentage of correct chunks in top parse improved from 78.42% to 88.26%. The percentage of correct parses in top position improved from 44.35% to 54.82%.

Chunkers are usually evaluated just for the percentage of correct chunks they produce, not for the correctness of the complete parse (chunk sequence) for the whole sentence. We have placed greater demands on ourselves and we expect our parser to produce optimal chunk sequence for the whole sentence. No word can be left out and there can be no overlaps either. Further, we produce all (or top few) combinations and that too in hopefully a best first order. Since we are aiming at chunks that correspond to answers to questions that can be asked of the given sentence, the very nature of the chunking task here is more semantic and hence more demanding. More over, we have used a fairly fine grained tag set with more than 70 tags. The data we have started with, namely the BNC POS tagged corpus, has tagging errors, multiple tags are given in many cases, some words are not tagged, and the tag set had to be extended. Given these factors, the performance we are able to achieve both in terms of percentage of correct chunks in the top parse and rank of the fully correct parse is very encouraging. We are demanding perfect match with manually parsed sentences and in most cases we have observed that the top parse is nearly correct. For example, just could be just a NN1/NP0 error in one of the chunks - not a serious problem for many applications.

The UCSG parser developed for English is a wide coverage shallow parsing system. The system has been built and tested on very large data sets, covering a wide variety of texts, giving us confidence that the system will perform well on new, unseen texts. The system is general and not domain specific, but we can adapt and fine tune for any specific domain as and when needed. We are confident that wide coverage and robust shallow parsing systems can be developed using the UCSG architecture for other languages of the world as well. We plan to continue our work on English parsing while we also start our work on Telugu.

6.2 Major Claims and Achievements

1. *It is possible to develop wide coverage partial parsing systems for Natural Languages in a reasonable amount of time without need for a parsed training corpus to start with* [75, 97]. This has been demonstrated for the case of English in this work.

2. *This can be achieved by a judicious combination of linguistic and statistical techniques.* In this thesis we have shown that a finite state grammar at chunk level combined with HMMs form a good combination. An architecture for wide coverage partial parsing has been proposed.

3. *Finite State Grammars with very high Recall can be built with relative ease at chunk level.* Finite State Grammars are easy to understand and visualize. Recognition with Finite State Grammars is computationally efficient - linear time algorithms exist. In this work, we have seen that very high Recall is achievable for English.

4. *The chunks produced by the UCSG system are somewhat more semantically oriented and closer to what is expected in a full syntactic parsing.* See examples at the end.

5. *Chunk level HMMs can be developed from a large POS Tagged corpus using the Finite State Grammar-Parser, without need for a parsed training corpus to start with.* Here we have developed HMMs from the British National Corpus of English and demonstrated their effectiveness.

6. *HMMs are used only for rating and ranking the chunks already obtained from the Finite State Grammar-Parser, not for recognizing chunks per se.* Since the chunks are already available and POS tags are also known for each word, even the Forward/Backward algorithm is not required and *evaluation can be done in linear time.*

7. *HMMs can produce good ranking, tending to push the correct chunks to positions near the top.* Good Mean Rank Scores have been achieved.

8. *HMM parameters can be refined by bootstrapping.* This has been demonstrated successfully in our bootstrapping experiments.

9. *A variety of Best First Search strategies can be employed to obtain globally best chunk sequences or parses for a given sentence.* In this work we have shown how best first search strategy can be used for this purpose. The time and space complexity for best first strategy is exponential in nature as branching factor and sentence length increases. Hence, we have also proposed a modified beam search strategy to improve the efficiency. Here we may lose some of the correct parses if there is heavy pruning - the results depend on size of the beam. Some work has also been done in applying the A* best first search algorithm. There is scope for incorporating more linguistic constraints here.

10. *It is possible to develop high quality (manually checked) parsed corpora using the system.* A 4000 sentence manually checked parsed corpus has been developed and used for development, bootstrapping, as also for testing and evaluation.

11. Large scale POS tagged corpora are available, or can be easily developed, for other languages of the world, including Indian languages. Indian languages are characterized by free word order and rich morphology. Nevertheless, words within chunks are order specific and thus Finite State Grammars at chunk level will not be much different. *The overall architecture should therefore be of interest in developing wide coverage partial parsing systems for Indian languages as well.*

12. *A wide coverage dictionary for English including frequency of occurrence of each word in each Tag has been developed and found to be useful for chunk generation as well as rating and ranking using HMMs.*

13. *A Decision Tree solution for the sentence boundary detection problem has been developed and shown to give very good performance.*

14. *A Language Identification system from small text samples for pair-wise language identification among 9 major Indian languages has also been developed* using multiple linear regression as a two-class classification model. Good performance has been obtained.

15. *A variety of statistical analyses have been carried out on a nearly 40 Million word text corpus of Telugu. The results should be useful for further work on Telugu.*

Chapter 7

Further Work

1. The dictionary has a very good coverage and has been found to be very useful both for recognizing chunks and for rating and ranking chunks using HMMs. Closed class words have been manually checked but open class words need to be checked and cleaned up further by removing unwanted tags. For example, 33,000 words are tagged as both NN1 and NP0, multiplying the chunks produced. Dictionary refinement can be done by manual checking, by cross-validating with other dictionaries, or by developing parsed corpora using the parsing system developed and working backwards.

2. It has been seen that performance of the parser can be significantly improved by incorporating a POS tagger. Since we already have chunk level HMMs as also chunk sequence statistics from the parsed corpus, it would be interesting to see how these can be exploited for developing a high performance POS tagger with matching tag set.

3. A variety of machine learning strategies as also sentence level linguistic constraints can be incorporated to further improve the chunk sequence selection.

4. Using the system larger parsed corpora can be developed and further bootstrapping experiments can be carried out.

5. The partial parsing system is currently producing only chunk sequences as parse output. Thematic role assignment (such as subject and object) should be included. Further enhancements to handle clause structure would take it closer to a full syntactic parsing system.

6. The architecture can be applied for developing wide coverage partial parsing systems for Indian languages. Of particular interest would be the case

of Telugu since fairly large text corpora as also a variety of resources and tools including dictionaries and morphological analyzers are already available.

7. Application of parse results for other NLP tasks such as Machine Translation, Text Categorization, Information Retrieval, Information Extraction and Automatic Summarization could be explored.

Bibliography

[1] S. Abney. *Corpus-Based Methods in Language and Speech*, chapter Part-of-Speech Tagging and Partial Parsing. Kluwer Academic Publishers, Dordrecht, 1996.

[2] Steven Abney. Partial parsing. In *ANLP-94*, 1994.

[3] Steven Abney. Partial parsing via finite-state cascades. In *Workshop on Robust Parsing, 8th European Summer School in Logic, Language and Information*, pages 8–15, Prag, 1996.

[4] Steven P. Abney. *Parsing by Chunks.* Kluwer, principle-based parsing: computation and psycholinguistics edition, 1991.

[5] K. Ajdukiewicz. Die syntaktische konnexit. *Polish Logic*, 1(27):207–231, 1935.

[6] I. Aldezabal, M. Aranzabe, A. Atutxa, K. Gojenola, M. Oronoz, and K. Sarasola. Application of finite-state transducers to the acquisition of verb subcategorization information. *Natural Language Engineering*, 9(1):39–48, 2003.

[7] Ulrike Baldewein, Katrin Erk, Sebastian Padó, and Detlef Prescher. Semantic role labelling with chunk sequences. In *Proceedings of CoNLL-2004*, pages 98–101. Boston, MA, USA, 2004.

[8] Y. Bar-Hillel. On syntactical categories. *Journal of Symbolic Logic*, 15:1–16, 1950.

[9] Roberto Basili, Maria Teresa Pazienza, and Fabio Massimo Zanzotto. Lexicalizing a shallow parser. In *Proceedings of TALN*, 1999.

[10] Beareau of Indian Standards. Indian script code for information interchange - ISCII. In *IS 13194: 1991*, New Delhi, India, 1991.

[11] Akshar Bharati, Medhavi Bhatia, Vineet Chaitanya, and Rajeev Sangal. Paninian grammar applied to english. *Computer Science and Informatics*, Special issue on Natural Language Processing and Machine Learning, 1995.

[12] Akshar Bharati, Sriram Venkatapathy, and Prashanth Reddy. Inferring semantic roles using sub-categorization frames and maximum entropy model. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 165–168, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[13] R. Bod and R. Scha. Data-oriented language processing: An overview. Technical Report Technical Report LP-96-13, Institute for Logic, Language and Computation, University of Amsterdam, 1996.

[14] Rens Bod. An empirical evaluation of lfg-dop. In *Proceedings COLING-2000*, Saarbr cken, Germany, 2000.

[15] Thorsten Brants. Cascaded markov models. In *Proceedings of 9th Conference of the European Chapter of the Association for Computational Linguistics (EACL-99)*, Bergen, Norway, 1999.

[16] L. Burnard. The users reference guide for the British National Corpus. Oxford University Computing Services, Oxford, 2000.

[17] Andrew J. Carlson, Chad M. Cumby, Jeff L. Rosen, and Dan Roth. Snow user's guide. Technical Report UIUC Tech report, UIUC Tech report, 1999.

[18] Xavier Carreras and Lluís Màrques. Introduction to the conll-2004 shared task: Semantic role labeling. In *Proceedings of CoNLL-2004*, pages 89–97. Boston, MA, USA, 2004.

[19] Xavier Carreras, Lluís Màrquez, and Grzegorz Chrupala. Hierarchical recognition of propositional arguments with perceptrons. In *Proceedings of CoNLL-2004*, pages 106–09. Boston, MA, USA, 2004.

[20] Xavier Carreras and Luís Màrquez. Boosting trees for clause splitting. In Walter Daelemans and Rémi Zajac, editors, *Proceedings of CoNLL-2001*, pages 73–75. Toulouse, France, 2001.

[21] E. Charniak, S. Goldwater, and M. Johnson. Edge-based best-first chart parsing. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 127–133, 1998.

[22] Eugene Charniak. Statistical parsing with a context-free grammar and word statistics. In *AAAI/IAAI*, pages 598–603, 1997.

[23] Eugene Charniak. A maximum-entropy-inspired parser. Technical report, 2000.

[24] Chi. Statistical properties of probabilistic context-free grammars. In *Computational Linguistics, MIT Press for the Association for Computational Linguistics*, volume 25, 1999.

[25] Noam Chomsky. *Lectures on Government and Binding.* Foris, Dordrecht, 1981.

[26] FABIO CIRAVEGNA and ALBERTO LAVELLI. Full parsing approximation for information extraction via finite-state cascades. *Natural Language Engineering*, 8(2):145–165, 2002.

[27] Cognitive Computation Group Shallow Parser. `http://l2r.cs.uiuc.edu/~cogcomp/eoh/index.html`.

[28] Trevor Cohn and Philip Blunsom. Semantic role labelling with tree conditional random fields. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 169–172, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[29] M. Collins, J. Hajic, L. Ramshaw, and C. Tillmann. A statistical parser for czech. In *Proceedings of the 37th Annual Meeting of the ACL*, College Park, Maryland, 1999.

[30] Michael John Collins. A new statistical parser based on bigram lexical dependencies. In Arivind Joshi and Martha Palmer, editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 184–191, San Francisco, 1996. Morgan Kaufmann Publishers.

[31] V.J. Cook. *Chomksy's Universal Grammar: An Introduction.* Blackwell, 1988.

[32] Walter Daelemans, Jakub Zavrel, Peter Berck, and Steven Gillis. Memory-based shallow parsing. In *Proceedings of CoNLL-99*, Bergen, Norway, 1999.

[33] Robert Dale, Hermann Moisl, and Harold Somers. *Handbook of Natural Language Processing.* Marcel Dekker, New York, 2000.

[34] Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. Generating typed dependency parses from phrase structure. In *Proceedings of LREC-2006*, 2006.

[35] Hervé Déjean. Using allis for clausing. In Walter Daelemans and Rémi Zajac, editors, *Proceedings of CoNLL-2001*, pages 64–66. Toulouse, France, 2001.

[36] Hervi Dejean. Learning syntactic structures with xml. In *Proceedings of CoNLL-2000 and LLL-2000*, Lisbon, Portugal, 2000.

[37] Hervi Dejean. Learning rules and their exceptions. In *Journal of Machine Learning Research, volume 2*, pages 669–693, 2002.

[38] Christy Doran, Dania Egedi, Beth Ann Hockey, B. Srinivas, and Martin Zaidel. XTAG system – a wide coverage grammar for english. In *Proceedings of the 15th. International Conference on Computational Linguistics (COLING 94)*, volume II, pages 922–928, Kyoto, Japan, 1994.

[39] Christy Doran, Dania Egedi, Beth Ann Hockey, B. Srinivas, and Martin Zaidel. XTAG system – a wide coverage grammar for english. In *Proceedings of the 15th. International Conference on Computational Linguistics (COLING 94)*, volume II, pages 922–928, Kyoto, Japan, 1994.

[40] E. Appelt Douglas, Jerry R. Hobbs, John Bear, David Israel, and Mabry Tyson. Fastus: A finite-state processor for information extraction from real-world text'. In *Proceedings of IJCAI-93*, Chambery, France, 1993.

[41] Rakesh Dugad and U. B. Desai. A tutorial on hidden markov models. Technical Report SPANN-96.1, IIT Bombay, 1996.

[42] E. Black et al. Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of DARPA Speech and Natural Language Workshop*, 1992.

[43] Charles J. Fillmore. *Universals in Linguistic Theory*, chapter The case for case. Holt and Rinehart and Winston, 1968.

[44] G Gazdar, C.S. Mellish, G Pullum, and I. Sag. *Genaralized Phrase Structure Grammar*. Oxford:Basil Blackwell, 1985.

[45] Gee, James Paul, and Franis Grosjean. Performance structures: A psycholinguistic and linguistic appraisal. *Cognitive Psychology*, 15:411–458, 1983.

[46] Daniel Gildea. Corpus variation and parser performance. In *Proceedings of Conference on Empirical Methods in Natural Language*, 2001.

[47] Leroy Gondy, Chen Hsinchun, and Martinez Jesse. A shallow parser based on closed-class words to capture relations in biomedical text. In *Journal of Biomedical Informatics 36*, pages 145–158, 2003.

[48] J. Goodman. Probabilistic feature grammars. 1997.

[49] G. Grefenstette. Light parsing as finite state filtering. In *Workshop on Extended finite state models of language*, Budapest, Hungary, 1996.

[50] Dennis Grinberg, John Lafferty, and Daniel Sleator. A robust parsing algorithm for LINK grammars. Technical Report CMU-CS-TR-95-125, CMU,Pittsburgh, PA, 1995.

[51] The XTAG Research Group. A lexicalized tree adjoining grammar for english. IRCS Report 95-03, Univ. of Pennsylvania, 1995.

[52] Kadri Hacioglu, Sameer Pradhan, Wayne Ward, James H. Martin, and Daniel Jurafsky. Semantic role labeling by tagging syntactic chunks. In *Proceedings of CoNLL-2004*, pages 110–113. Boston, MA, USA, 2004.

[53] Aria Haghighi, Kristina Toutanova, and Christopher Manning. A joint model for semantic role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 173–176, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[54] James Hammerton. Clause identification with long short-term memory. In Walter Daelemans and Rémi Zajac, editors, *Proceedings of CoNLL-2001*, pages 61–63. Toulouse, France, 2001.

[55] Derrick Higgins. A transformation-based approach to argument labeling. In *Proceedings of CoNLL-2004*, pages 114–117. Boston, MA, USA, 2004.

[56] A. S. Hornby. *Guide to Patterns and Usage in English.* Oxford University Press, 1975.

[57] Timo Jarvinen and Pasi Tapanainen. A dependency parser for english. Technical Report TR-1, Department of General Linguistics, University of Helsinki, 1997.

[58] Bresnan Joan. *Lexical-Functional Syntax.* BlackWell, 2001.

[59] Christer Johansson. A context sensitive maximum likelihood approach to chunking. In *Proceedings of CoNLL-2000 and LLL-2000*, Lisbon, Portugal, 2000.

[60] Richard Johansson and Pierre Nugues. Sparse bayesian classification of predicate arguments. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 177–180, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[61] A. K. Joshi and Schabes. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, pages 69 – 124, San Francisco, 1997. Springer.

[62] Aravind K Joshi. Tree Adjoining Grammar: How much Context-Sensitivity is Required to Provide reasonable Structural Descriptions. In L Karttunen D Dowty and A Zwicky, editors, *Natural Language Parsing.* Cambridge University Press, Cambridge, UK, 1985.

[63] Aravind K Joshi. Tree Adjoining Grammar: How much Context-Sensitivity is Required to Provide reasonable Structural Descriptions. In L Karttunen D Dowty and A Zwicky, editors, *Natural Language Parsing.* Cambridge University Press, Cambridge, UK, 1985.

[64] Daniel Jurafsky and James H. Martin. *Speech and Language Processing-An introduction to Natural Language Processing, Computational Linguistics and Speech Recognition.* Pearson Education, 2002.

[65] Ronald M. Kaplan. The formal architecture of lexical-functional grammar. In *Proceedings of ROCLING II*, pages 1–18, 1989.

[66] Ronald M Kaplan and Joan Bresnan. *Lexical-Functional Grammar: A Formal System of Grammatical Representation.* MIT Press, Cambridge, Massachusetts, 1982.

[67] M. Kay. Functional unification grammar:a formalism for machine translation. In *Proceedings of the 10th International Conference on Computational Linguistics/22nd annual meeting of ACL*, pages 75–78, CA, November 1984.

[68] Dan Klein and Christopher D. Manning. An $O(n^3)$ agenda-based chart parser for arbitrary probabilistic context-free grammars. Technical Report dbpubs/2001-16, Stanford University, 2001.

[69] Dan Klein and Christopher D. Manning. Fast exact inference with a factored model for natural language parsing. In *Advances in Neural Information Processing Systems 15 (NIPS 2002)*, 2002.

[70] Dan Klein and Christopher D. Manning. A* parsing: Fast exact viterbi parse selection. In *Proceedings of HLT-NAACL 03*, 2003.

[71] Rob Koeling. Chunking with maximum entropy models. In *Proceedings of CoNLL-2000 and LLL-2000*, Lisbon, Portugal, 2000.

[72] Peter Koomen, Vasin Punyakanok, Dan Roth, and Wen-tau Yih. Generalized inference with multiple semantic role labeling systems. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 181–184, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[73] Beata Kouchnir. A memory-based approach for semantic role labeling. In *Proceedings of CoNLL-2004*, pages 118–121. Boston, MA, USA, 2004.

[74] Taku Kudoh and Yuji Matsumoto. Use of support vector learning for chunk identification. In *Proceedings of CoNLL-2000 and LLL-2000*, Lisbon, Portugal, 2000.

[75] G Bharadwaja Kumar and Kavi Narayana Murthy. Ucsg shallow parser. *Proceedings of CICLING 2006, LNCS*, 3878:156–167, 2006.

[76] G Bharadwaja Kumar, Kavi Narayana Murthy, and B B Chaudhuri. Statistical analyses of telugu text corpora. *International journal of Dravidian linguistics*, 36(2), 2007.

[77] Xin Li and Dan Roth. Exploring evidence for shallow parsing. In *Proceedings of the Annual Conference on Computational Natural Language Learning*, 2001.

[78] Joon-Ho Lim, Young-Sook Hwang, So-Young Park, and Hae-Chang Rim. Semantic role labeling using maximum entropy model. In *Proceedings of CoNLL-2004*, pages 122–125. Boston, MA, USA, 2004.

[79] Chi-San Lin and Tony C. Smith. Semantic role labeling via consensus in pattern-matching. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 185–188, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[80] Dekang Lin. Principar–an efficient, broad-coverage, principle-based parser. In *Proceedings of COLING-94*, pages 42–48, Kyoto, Japan, 1994.

[81] Dekang Lin. Dependency-based evaluation of minipar. In *Workshop on the Evaluation of Parsing Systems*, Granada, Spain, 1998.

[82] Ting Liu, Wanxiang Che, Sheng Li, Yuxuan Hu, and Huaijun Liu. Semantic role labeling system using maximum entropy classifier. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 189–192, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[83] D. Magerman. Statistical decision-tree models for parsing. In *Proc. of the 33 ACL*, pages 276–283, Cambridge, MA, 1995.

[84] Lluís Màrquez, Pere Comas, Jesús Giménez, and Neus Català. Semantic role labeling as sequential tagging. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 193–196, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[85] B Megyesi. Shallow parsing with pos taggers and linguistic features. In *Journal of Machine Learning Research, volume 2*, pages 639–668, 2002.

[86] Memory Based Shallow Parser. `http://ilk.uvt.nl/cgi-bin/tstchunk/demo.pl`.

[87] Collins Michael. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35 Annual Meeting of the ACL (jointly with the 8th Conference of the EACL)*, 1997.

[88] A. Mikheev. A knowledge-free method for capitalized word disambiguation. In *Proceedings of the 37th Annual Meeting of the ACL*, pages 159–166, University of California,Maryland., 1999.

[89] Tomohiro Mitsumori, Masaki Murata, Yasushi Fukuda, Kouichi Doi, and Hirohumi Doi. Semantic role labeling using support vector machines. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 197–200, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[90] Antonio Molina and Ferran Pla. Clause detection using hmm. In Walter Daelemans and Rémi Zajac, editors, *Proceedings of CoNLL-2001*, pages 70–72. Toulouse, France, 2001.

[91] Antonio Molina and Ferran Pla. Shallow parsing using specialized hmms. In *Journal of Machine Learning Research, volume 2*, pages 595–613, 2002.

[92] Alessandro Moschitti, Ana-Maria Giuglea, Bonaventura Coppola, and Roberto Basili. Hierarchical semantic role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 201–204, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[93] K. Narayana Murthy. *Universal Clause Structure Grammar*. PhD Thesis, University of Hyderabad, 1995.

[94] K. Narayana Murthy. UCSG and Machine Aided Translation from English to Kannada. In *Indo-french Symposium on Natural Language Processing*, 1997.

[95] K. Narayana Murthy. Universal Clause Structure Grammar and the Syntax of Relatively Free Word Order Languages. *South Asian Language Review*, VII(1), Jan 1997.

[96] Kavi Narayana Murthy. *Natural Language Processing - an Information Access Perspective*. Ess Ess Publications, New Delhi, India, 2006.

[97] Kavi Narayana Murthy and G. Bharadwaja Kumar. UCSG: A Hybrid Architecture for Wide Coverage Syntactic Parsing. Communicated.

[98] Kavi Narayana Murthy and G. Bharadwaja Kumar. Language identification from small text samples. *Journal of Quantitative Linguistics*, 13(1):57–80, 2006.

[99] K. Nagesh. Towards a robust shallow parser. Masters thesis, Department of Computer and Information Sciences, University of Hyderabad, 2004.

[100] N.Chomsky. *A minimalist program for linguistic theory.* MIT Press, the view from building 20: essays in honor of sylvain bromberger edition, 1993.

[101] Atul Negi, Kavi Narayana Murthy, and Chakravarthy Bhagvati. Foundational issues of document engineering in indian scripts and a case study in telugu. *Vivek*, 16(2):2–7, 2006.

[102] Constantin Orasan. A hybrid method for clause splitting in unrestricted english texts. In *Proceedings of ACIDCA'2000*, 2000.

[103] Miles Osborne. Shallow parsing as part-of-speech tagging. In *Proceedings of CoNLL-2000 and LLL-2000*, Lisbon, Portugal, 2000.

[104] Miles Osborne. Shallow parsing using noisy and non-stationary training material. In *Journal of Machine Learning Research, volume 2*, pages 695–719, 2002.

[105] Ozgencil, Necati Ercan, and Nancy McCracken. Semantic role labeling using libSVM. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 205–208, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[106] Harris V. Papageorgiou. Clause recognition in the framework of alignment. In *Recent Advances in Natural Language Processing.* John Benjamins, 1997.

[107] Kyung-Mi Park, Young-Sook Hwang, and Hae-Chang Rim. Two-phase semantic role labeling based on support vector machines. In *Proceedings of CoNLL-2004*, pages 126–129. Boston, MA, USA, 2004.

[108] Kyung-Mi Park and Hae-Chang Rim. Maximum entropy based semantic role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 209–212, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[109] Jon D. Patrick and Ishaan Goyal. Boosted decision graphs for nlp learning tasks. In Walter Daelemans and Rémi Zajac, editors, *Proceedings of CoNLL-2001*, pages 58–60. Toulouse, France, 2001.

[110] Ferran Pla, Antonio Molina, and Natividad Prieto. Improving chunking by means of lexical-contextual information in statistical language models. In *Proceedings of CoNLL-2000 and LLL-2000*, Lisbon, Portugal, 2000.

[111] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press and CSLI Publications, Chicago, Illinois, 1994.

[112] Simone Paolo Ponzetto and Michael Strube. Semantic role labeling using lexical statistical information. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 213–216, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[113] Sameer Pradhan, Kadri Hacioglu, Wayne Ward, James H. Martin, and Daniel Jurafsky. Semantic role chunking combining complementary syntactic views. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 217–220, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[114] Vasin Punyakanok, Dan Roth, Wen tau Yih, Dav Zimak, and Yuancheng Tu. Semantic role labeling via generalized inference over classifiers. In *Proceedings of CoNLL-2004*, pages 130–133. Boston, MA, USA, 2004.

[115] Georgiana Puscasu. A multilingual method for clause splitting. In *Proceedings of CLUK 2004*, pages 199–206, Birmingham, UK, 2004.

[116] J. Ross Quinlan. *C4.5 Programs for Machine Learning*. Morgan Kaufmann, 1993.

[117] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, pages 257–285, 1989.

[118] Lance A. Ramshaw and Mitchell P. Marcus. Text chunking using transformation-based learning. In *Proceedings of the Third Workshop on Very Large Corpora*, pages 82–94, Cambridge, MA, USA, 1995.

[119] Adwait Ratnaparkhi. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34:151–175, 1999.

[120] Reuters news group. Reuters corpus. In *Reuters corpus Volume 1, English language, Format version 1*, 2000.

[121] Brian Roark. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27(2):249–276, 2001.

[122] T.G. Rose, M. Stevenson, and M. Whitehead. The reuters corpus volume 1 - from yesterday's news to tomorrow's language resources. In *Proceedings of the Third International Conference on Language Resources and Evaluation*, Las Palmas de Gran Canaria, 2002.

[123] Geoffrey Sampson. *English For The Computer*. Clarendon Press (the scholarly imprint of Oxford University Press), 1995.

[124] E. F. Tjong Kim Sang and S. Buchholz. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of CoNLL-2000 and LLL-2000*, pages 127–132, Lisbon, Portugal, 2000.

[125] Erik F. Tjong Kim Sang. Text chunking by system combination. In *Proceedings of CoNLL-2000 and LLL-2000*, Lisbon, Portugal, 2000.

[126] Erik F. Tjong Kim Sang. Memory-based shallow parsing. In *Journal of Machine Learning Research, volume 2*, pages 559–594, 2002.

[127] Erik F. Tjong Kim Sang and HervDejean. Introduction to the conll-2001 shared task: Clause identification. In *Proceedings of CoNLL-2001*, Toulouse, France, 2001.

[128] Tjong Kim Sang and Erik F. Memory-based clause identification. In Walter Daelemans and Rémi Zajac, editors, *Proceedings of CoNLL-2001*, pages 67–69. Toulouse, France, 2001.

[129] Gerold Schneider, James Dowdall, and Fabio Rinaldi. A robust and hybrid deep-linguistic theory applied to large-scale parsing. In *Recent advances in DG workshop, Coling 2004*, Geneva, 2004.

[130] Peter Sells. *Lectures on Contemporary Syntactic Theories*. Center for the Study of Language and Information, Lecture Notes, No. 3 edition, 1968.

[131] F. Sha and F. Pereira. Shallow parsing with conditional random fields. Technical Report CIS TR MS-CIS-02-35, 2003.

[132] K Vijay Shankar and A. K. Joshi. *Unification-based Grammars*, chapter Unification-based Tree Adjoining Grammars. MIT Press, 1991.

[133] Libin Shen and Aravind K. Joshi. A snow based supertagger with application to np chunking. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 505–512, Japan, 2003. Association for Computational Linguistics.

[134] Daniel DK Sleator and Davy Temperley. Parsing english with a link grammar. In *Third International Workshop on Parsing Technologies*, 1993.

[135] M. steedman. Categorial grammar. *Lingua*, 90:221–258, 1993.

[136] Mihai Surdeanu and Jordi Turmo. Semantic role labeling using complete syntactic analysis. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 221–224, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[137] Charles Sutton and Andrew McCallum. Joint parsing and semantic role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 225–228, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[138] Lucien Tesniere. *Elements de syntax structural.* Klincksieck, Paris, 1959.

[139] Erik Tjong Kim Sang, Sander Canisius, Antal van den Bosch, and Toine Bogers. Applying spelling error correction techniques for improving semantic role labelling. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 229–232, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[140] Tzong-Han Tsai, Chia-Wei Wu, Yu-Chun Lin, and Wen-Lian Hsu. Exploiting full parsing information to label semantic roles using an ensemble of ME and SVM via integer linear programming. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 233–236, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[141] Paul E. Utgoff. *Incremental Induction of Decision Trees.* Kluwer Academic Publishers, 1989.

[142] van den Bosch Antal, Sander Canisius, Walter Daelemans, Iris Hendrickx, and Erik Tjong Kim Sang. Memory-based semantic role labeling: Optimizing features, algorithm, and output. In *Proceedings of CoNLL-2004*, pages 102–105. Boston, MA, USA, 2004.

[143] Hans van Halteren. Chunking with wpdv models. In *Proceedings of CoNLL-2000 and LLL-2000*, Lisbon, Portugal, 2000.

[144] Jorn Veenstra and Antal van den Bosch. Single-classifier memory-based phrase chunking. In *Proceedings of CoNLL-2000 and LLL-2000*, Lisbon, Portugal, 2000.

[145] Marc Vilain and David Day. Phrase parsing with rule sequence processors: an application to the shared conll task. In *Proceedings of CoNLL-2000 and LLL-2000*, Lisbon, Portugal, 2000.

[146] Wolfgang Wahlster. *Verbmobil: Foundations of Speech-to-Speech Translation.* Springer, 2000.

[147] Hanna M. Wallach. Conditional random fields: An introduction. Technical Report MS-CIS-04-21, Department of Computer and Information Science, University of Pennsylvania, 2004.

[148] Amy Weinberg. *A Minimalist Theory of Human Sentence Processing.* MIT Press, working minimalism edition, 1999.

[149] Ken Williams, Christopher Dozier, and Andrew McCulloh. Learning transformation rules for semantic role labeling. In *Proceedings of CoNLL-2004*, pages 134–137. Boston, MA, USA, 2004.

[150] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann, 1999.

[151] Szu-ting Yi and Martha Palmer. The integration of syntactic parsing and semantic role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 237–240, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[152] Keh Kok Yong and Christian Huyck. Robust parsing using plink. In *Proceedings of the 1st International Conference on Vision, Information and Parallel Processing for Industrial Automation (ROVISP),*, pages 269–275, 2003.

[153] Tong Zhang, Fred Damerau, and David Johnson. Text chunking based on a generalization of winnow. In *Journal of Machine Learning Research, volume 2*, pages 615–637, 2002.

[154] GuoDong Zhou, Jian Su, and TongGuan Tey. Hybrid text chunking. In *Proceedings of CoNLL-2000 and LLL-2000*, Lisbon, Portugal, 2000.

Appendix A

Tag set used in UCSG Grammar

Most of the tags used for our work are taken from the BNC C5 tagset[16]. We have extended the tag set wherever it is necessary. We first give the tags from C5:

1. AJ0 Adjective (general or positive) (e.g. good, old, beautiful)
2. AJC Comparative adjective (e.g. better, older)
3. AJS Superlative adjective (e.g. best, oldest)
4. AT0 Article (e.g. the, a, an, no)
5. AV0 General adverb: an adverb not subclassified as AVP or AVQ (see below) (e.g. often, well, longer)
6. AVP Adverb particle (e.g. up, off, out)
7. AVQ Wh-adverb (e.g. when, where, how, why, wherever)
8. CJC Coordinating conjunction (e.g. and, or, but)
9. CJS Subordinating conjunction (e.g. although, when)
10. CJT The subordinating conjunction that
11. CRD Cardinal number (e.g. one, 3, fifty-five, 3609)
12. DPS Possessive determiner (e.g. your, their, his)
13. DT0 General determiner: i.e. a determiner which is not a DTQ (e.g Any, Both, Each)
14. DTQ Wh-determiner (e.g. which, what, whose, whichever)

15. EX0 Existential there
16. ITJ Interjection or other isolate (e.g. oh, yes, mhm, wow)
17. NN0 Common noun, neutral for number (e.g. aircraft, data, committee)
18. NN1 Singular common noun (e.g. pencil, goose, time, revelation)
19. NN2 Plural common noun (e.g. pencils, geese, times, revelations)
20. NP0 Proper noun (e.g. London, Michael, Mars, IBM)
21. ORD Ordinal numeral (e.g. first, sixth, 77th, last)
22. PNI Indefinite pronoun (e.g. none, everything, one [as pronoun], nobody)
23. PNP Personal pronoun (e.g. all, both, any)[Note: Here we have included some pronouns which are considered as only determiners in BNC tagset.]
24. PNQ Wh-pronoun (e.g. who, whoever, whom)
25. PNX Reflexive Pronoun (e.g. myself, yourself, itself, ourselves)
26. PRF The preposition of.
27. PRP Preposition (except for of) (e.g. about, at, in, on, on behalf of, with)
28. PUL Punctuation: left bracket - i.e. (or [
29. PUN Punctuation: general separating mark - i.e. . , ! , : ; - or ?
30. PUQ Punctuation: quotation mark - i.e. ' or "
31. PUR Punctuation: right bracket - i.e.) or]
32. TO0 Infinitive marker to
33. VBB The present tense forms of the verb BE, except for is, 's: i.e. am, are, 'm, 're and be [subjunctive or imperative]
34. VBD The past tense forms of the verb BE: was and were
35. VBG The -ing form of the verb BE: being
36. VBN The past participle form of the verb BE: been
37. VBZ The -s form of the verb BE: is, 's

38. VDB The finite base form of the verb BE: do
39. VDD The past tense form of the verb DO: did
40. VDG The -ing form of the verb DO: doing
41. VDN The past participle form of the verb DO: done
42. VDZ The -s form of the verb DO: does, 's
43. VHB The finite base form of the verb HAVE: have, 've
44. VHD The past tense form of the verb HAVE: had, 'd
45. VHG The -ing form of the verb HAVE: having
46. VHN The past participle form of the verb HAVE: had
47. VHZ The -s form of the verb HAVE: has, 's
48. VM0 Modal auxiliary verb (e.g. will, would, can, could, 'll, 'd)
49. VVB The finite base form of lexical verbs (e.g. forget, send, live, return) [Including the imperative and present subjunctive]
50. VVD The past tense form of lexical verbs (e.g. forgot, sent, lived, returned)
51. VVG The -ing form of lexical verbs (e.g. forgetting, sending, living, returning)
52. VVN The past participle form of lexical verbs (e.g. forgotten, sent, lived, returned)
53. VVZ The -s form of lexical verbs (e.g. forgets, sends, lives, returns)
54. XX0 The negative particle not or n't

A.1 Extensions to the C5 tag set

1. DTP Predeterminers(all, both, such, what, many, half, quite)
2. PRN Preposition that can combine with conjunction to form a conjunction group(e.g. in which, by whom, below which, in that)
3. PNN Pronoun Nominative (e.g I, he, they)

4. PNA Pronoun Accusative (e.g him, her, them)
5. PPS Pronoun Possessive (e.g mine, hers, his, its)
6. PNC Pronouns which can be both nominative and accusative (e.g. you, it)
7. NAV Post Nominal Adverbs those can be included into noun groups (alone, also, each, only, now, respectively, therefore, therein, too, ago, else, onwards)
8. NPS Noun Possessive (e.g. Peter's, Rama's)
9. AJBLE Adjectives ending with 'ble' (The bills payable)
10. VS1 Special verbs (e.g. used, ought)
11. VS2 Special verbs (e.g. need, dare)
12. VS3 Special verbs (e.g. go, keep, went)
13. SW1 The word 'on' treated as Special word if it comes in verb groups (e.g. keep on, went on)
14. SW2 The word 'about' treated as Special word if it comes in verb groups (e.g. is about to go)
15. AS1 adverb 'how' tagged as AS1 to recognize (how best, how many and how much) as a single question adverb
16. AS2 adverb 'as' is tagged as AS2 to recognize adverb and conjunctions like (as soon as, as long as, etc.)
17. CJR Relative conjunction (e.g. which, what, who)

Note 1: We have removed infinitive tags (VBI,VDI,VHI,VVI) from our tag set - they are same as the base forms of the verbs.

Note 2: The tag 'POS' in BNC tag set is changed to NPS. This modification is done because the genitive markers 's or ' are considered separate words in BNC corpus and tagged as POS. In our work we include the genitive markers with previous words and tag as NPS.

Note 3: Some tags like UNC, ZZ0 are not used in our grammar. These tags are also eliminated from our dictionary.

Appendix B

Finite State Grammar

Here, we have given finite state grammar for verb groups.

Table B.1: Finite State Grammar for Verb Groups

Start State	Corresponding C5 Tags	End State	Final State?
1	VVB VVD VVZ	2	vg
1	AV0	15	
15	VVB VVD VVZ	2	vg
1	VBB VBD VBZ	3	
15	VBB VBD VBZ	3	
3	AV0,AVQ,XX0	3	vg
3	VBG	11	
3	VVG,VHG,VDG,VVN,VDN	2	
3	SW2	5	
5	TO0	4	vg
1	VM0	4	vg
15	VM0	4	vg
4	VBB	3	vg
4	AV0,XX0	4	vg
4	VHB	6	vg
4	VVB,VDB	2	vg
1	VHB VHD VHZ	6	vg
15	VHB VHD VHZ	6	vg
6	TO0	8	vg
6	VBN	10	vg
6	VVN,VHN,VDN	2	vg
6	XX0,AV0	6	

Table B.2: Finite State Grammar for Verb Groups

Start State	Corresponding C5 Tags	End State	Final State?
1	VS2	7	vg
15	VS2	7	vg
7	XX0,AV0	7	vg
7	TO0	16	
7	VDB,VHB,VVB ,VBB	2	vg
1	VDB,VDD,VDZ	8	vg
15	VDB,VDD,VDZ	8	vg
8	VDB,VHB,VVB ,VBB	2	vg
8	VBB,VHB	11	
8	AV0,XX0	8	
1	VS1	9	
9	XX0	9	
9	TO0	16	vg
16	VDB,VHB,VVB,VBB	2	vg
16	VBB,VHB	11	
16	AV0	16	
10	AV0	10	vg
10	VBG	11	
10	VVN,VHN,VDN, VVG,VHG,VDG	2	vg
11	VVN,VHN,VDN	2	
11	AV0	11	
1	VS3	12	
15	VS3	12	
12	AV0	12	
12	SW1	13	
13	AV0	13	
13	VVG,VHG,VDG	2	vg
12	VVG,VHG,VDG	2	vg
1	VHG,VBG	14	
14	AV0	14	
14	VVN,VDN	2	
2	AVP,AV0	2	vg

Appendix C

Examples from Manually Parsed Corpus

Example 1

Tagged Sentence:

```
<AT0><a>##<ORD_NN1><second>##<VVB_NN1><phase>##
<PRN_PRF_AVP><of>##<VVB_NN1><research>##<CJR_PNQ_DTQ><which>##
<VBZ><is>##<AV0_AJ0><still>##<VBG_NN1><being>##<VVN_VVD><planned>##
<NP0_VM0_NN1><will>##<VVB_NN1><test>##<VVN_VVD><reformulated>##
<NN2><gasolines>##<PRN_PRP_AVP_SW1><on>##<AJC><newer>##
<NN1><engine>##<NN2><technologies>##<AV0_NAV><now>##<VBG_NN1><being>##
<VVN_VVD><developed>##<PRN_PRP_CJS_AVP><for>##<VVB_NN1><use>##
<PRN_PRP_AVP><in>##<CRD><1992>##<CJC><or>##<CRD><1993>##<NN2><cars>##
```

Manual Parse:

```
<ng>[<AT0><a>##<ORD><second>##<NN1><phase>##<PRF><of>##
<NN1><research>]</ng> <rel>[<CJR><which>]</rel>
<vg>[<VBZ><is>##<AV0><still>##<VBG><being>##<VVN><planned>]</vg>
<vg>[<VM0><will>##<VVB><test>]</vg>
<ng>[<VVN><reformulated>##<NN2><gasolines>]</ng>
<ng>[<PRP><on>##<AJC><newer>##<NN1><engine>##<NN2><technologies>]</ng>
<avg>[<AV0><now>]</avg> <vg>[<VBG><being>##<VVN><developed>]</vg>
<ng>[<PRP><for>##<NN1><use>]</ng>
<ng>[<PRP><in>##<CRD><1992>##<CJC><or>##<CRD><1993>##<NN2><cars>]</ng>
```

Example 2

Tagged Sentence:

```
<AT0><the>##<NN1><company>##<VHZ><has>##<AV0_NAV><also>##
<VVN_VVD><offered>##<AT0><a>##<VVB_NN1><plan>##<PRN_PRP_CJS_AVP><for>
##<VVB_NN1><commission>##<NN1><approval>##<CJT_DT0_PNP><that>##
<VM0><would>##<VVB><allow>##<NN1><gas>##<NN0><sales>##
<NN2><customers>##<PRN_TO0_PRP_AVP><to>##<AV0_DT0_PNP><either>##
<VVB><choose>##<AT0><a>##<VVN_VVD><fixed>##<AJ0_NN1><annual>##
<NN1><gas>##<VVN_VVB_VVD_NN1><cost>##<CJC><or>##<VVB_AJ0_NN1>
<level>##<NN1><gas>##<VVZ_NN2><costs>##<PRN_AV0_AVP_PRP><over>
##<AT0><a>##<NN1><period>##<PRN_PRF_AVP><of>##<DT0_PNP><several>##
<NN2><months>##
```

Manual Parse:

```
<ng>[<AT0><the>##<NN1><company>]</ng> <vg>[<VHZ><has>##<AV0><also>##
<VVN><offered>]</vg>
<ng>[<AT0><a>##<NN1><plan>##<PRP><for>##<NN1><commission>##
<NN1><approval>]</ng> <rel>[<CJT><that>]</rel>
<vg>[<VM0><would>##<VVB><allow>]</vg>
<ng>[<NN1><gas>##<NN0><sales>##<NN2><customers>]</ng>
<infg>[<TO0><to>##<AV0><either>##<VVB><choose>]</infg>
<ng>[<AT0><a>##<VVN><fixed>##<AJ0><annual>##<NN1><gas>##
<NN1><cost>]</ng> <coord>[<CJC><or>]</coord> <vg>[<VVB><level>]</vg>
<ng>[<NN1><gas>##<NN2><costs>]</ng> <ng>[<PRP><over>##<AT0><a>##
<NN1><period>##<PRF><of>##<DT0><several>##<NN2><months>]</ng>
```

Example 3

Tagged Sentence:

```
<NN1_VVB><cache>##<AJ0_NN1><capital>##<PUN><,>##
<VVN_VVD><based>##<PRN_PRP_AVP><in>##<NP0_NN1><bermuda>##
<PUN><,>##<NP0_VM0_NN1><will>##<VBB><be>##<AV0_VVB_AJ0_NN1>
<open>##<PRN_TO0_PRP_AVP><to>##<NP0><u.s.>##<NN2><investors>
##<PRN_AVP_PRP_AJ0><through>##<DPS_PPS><its>##
<NP0><Deleware>##<NN1><unit>##<NN1_VVB><cache>##
```

```
<AJ0_NN1><capital>##<NN1><lp>##<CJC><and>##<NP0_VM0_NN1>
<will>##<VHB><have>##<AT0><a>##<AJ0_NN1><minimum>##
<NN1><subscription>##<PRN_PRF_AVP><of>##<CRD><\$100,000>##
```

Manual Parse:

```
<ng>[<NN1><cache>##<NN1><capital>]</ng> <vgs>[<VVN><based>]</vgs>
<ng>[<PRP><in>##<NP0><bermuda>]</ng>
<vg>[<VM0><will>##<VBB><be>]</vg> <ajg>[<AJ0><open>]</ajg>
<ng>[<PRP><to>##<NP0><u.s.>##<NN2><investors>]</ng>
<ng>[<PRP><through>##<DPS><its>##<NP0><deleware>##<NN1><unit>]</ng>
<ng>[<NN1><cache>##<NN1><capital>]</ng> <ng>[<NN1><lp>]</ng>
<coord>[<CJC><and>]</coord> <vg>[<VM0><will>##<VHB><have>]</vg>
<ng>[<AT0><a>##<AJ0><minimum>##<NN1><subscription>##<PRF><of>##
<CRD><$100,000>]</ng>
```

Example 4

Tagged Sentence:

```
<PNN_CRD><i>##<VHB><have>##<VVN_VVD><found>##<AT0><the>
##<VVB_NN1><pen>##<CJR_PNQ_DTQ><which>##<PNN_CRD><i>##
<VHN_VHD><had>##<VVN_VVD><lost>##
```

Manual Parse:

```
<ng>[<PNN><i>]</ng> <vg>[<VHB><have>##<VVN><found>]</vg>
<ng>[<AT0><the>##<NN1><pen>]</ng> <rel>[<CJR><which>]</rel>
<ng>[<PNN><i>]</ng> <vg>[<VHD><had>##<VVN><lost>]</vg>
```

Appendix D

Publications

1. G Bharadwaja Kumar and Kavi Narayana Murthy, "*UCSG: A Wide Coverage Shallow Parsing System*", to appear in proceedings of IJCNLP, 7-12 January 2008.

2. G Bharadwaja Kumar and Kavi Narayana Murthy, "*UCSG Shallow Parser*", in proceedings of CICLing-2006, Lecture Notes in Computer Science, Volume 3878, pp 156-167, Springer-Verlag , 2006.

3. G. Bharadwaja Kumar, Kavi Narayana Murthy "*Towards a Robust Shallow Parser*", in proceedings of SIMPLE-2005, IIT Kharagpur, Feb 5-7, 2005.

4. Hla Hla Htay, G. Bharadwaja Kumar, Kavi Narayana Murthy "Constructing English- Myanmar Parallel Corpora", in proceedings of ICCA 2006: International Conference on Computer Applications, pp 231-238, Yangon, Myanmar, 23-24 February 2006.

5. G Bharadwaja Kumar, Kavi Narayana Murthy, B B Chaudhuri, "*Statistical Analysis of Telugu Text Corpora*", IJDL, Vol 36, No 2, June 2007.

6. Kavi Narayana Murthy, G Bharadwaja Kumar, "*Language Identification from Small Text Samples*", Journal of Quantitative Linguistics, Vol 13, No. 1, pp. 57-80, 2006.

7. G. Bharadwaja Kumar, Kavi Narayana Murthy "*Script Independent Language Identification in the Indian Context*", in proceedings of International Conference On Speech and Language Technology (ICLST-2004), TATA-McGraw-Hill publishers, pp 74-82, 2004.