CONTENTS

1. Introduction **1**

Chapter Overview *1*

1.1 What is Natural Language Processing (NLP) *1*

1.2 Origins of NLP *2*

1.3 Language and Knowledge *3*

UI 1.4 The Challenges of NLP *6*

1.5 Language and Grammar *8*

1.6 Processing Indian Languages *12*

1.7 NLP Applications *13*

1.8 Some Successful Early NLP Systems *15*

1.9 Information Retrieval *16*

2. Language Modelling **21**

Chapter Overview *21*

2.1 Introduction *21*

UI 2.2 Various Grammar-based Language Models *22*

2.3 Statistical Language Model *45*

3. Word Level Analysis **53**

Chapter Overview *53*

3.1 Introduction *53*

3.2 Regular Expressions *54*

3.3 Finite-State Automata *59*

UII 3.4 Morphological Parsing *63*

3.5 Spelling Error Detection and Correction *71*

3.6 Words and Word Classes *76*

3.7 Part-of-Speech Tagging *77*

4. Syntactic Analysis **92**

Chapter Overview *92*

4.1 Introduction *92*

UII 4.2 Context-Free Grammar *93*

4.3 Constituency *95*
4.4 Parsing *104*
4.5 Probabilistic Parsing *119*
4.6 Indian Languages *125*

5. Semantic Analysis 132
Chapter Overview *132*
5.1 Introduction *132*
5.2 Meaning Representation *134*
5.3 Lexical Semantics *145*
5.4 Ambiguity *151*
5.5 Word Sense Disambiguation *156*

6. Discourse Processing 179
Chapter Overview *179*
6.1 Introduction *179*
6.2 Cohesion *181*
6.3 Reference Resolution *185*
6.4 Discourse Coherence and Structure *196*

7. Natural Language Generation 209
Chapter Overview *209*
7.1 Introduction *209*
7.2 Architectures of NLG Systems *210*
7.3 Generation Tasks and Representations *213*
7.4 Applications of NLG *223*

8. Machine Translation 228
Chapter Overview *228*
8.1 Introduction *228*
8.2 Problems in Machine Translation *229*
8.3 Characteristics of Indian Languages *230*
8.4 Machine Translation Approaches *231*
8.5 Direct Machine Translation *232*
8.6 Rule-based Machine Translation *236*
8.7 Corpus-based Machine Translation *241*
8.8 Semantic or Knowledge-based MT systems *249*
8.9 Translation involving Indian Languages *250*

9. Information Retrieval–1 255
Chapter Overview *255*
9.1 Introduction *255*
9.2 Design Features of Information Retrieval systems *256*

CHAPTER OVERVIEW

This chapter gives an idea of natural language processing (NLP) and information retrieval (IR). Various levels of analysis involved in NLP along with the knowledge used by these levels of analysis are discussed. Some of the difficulties in analysing text and specific factors that make automatic processing of languages difficult are also touched upon. The chapter underlines the role of grammar in language processing and introduces transformational grammar. Indian languages differ a lot from English. These differences are clearly pointed out. Further, a number of NLP applications are introduced along with some of the early NLP systems. Towards the end, information retrieval is discussed.

.1 WHAT IS NATURAL LANGUAGE PROCESSING (NLP)

Language is the primary means of communication used by humans. It is the tool we use to express the greater part of our ideas and emotions. It shapes thought, has a structure, and carries meaning. Learning new concepts and expressing ideas through them is so natural that we hardly realize how we process natural language. But there must be some kind of representation in our mind, of the content of language. When we want to express a thought, this content helps represent language in real time. As children, we never learn a computational model of language, yet this is the first step in the automatic processing of languages. *Natural language processing* (NLP) is concerned with the development of computational models of aspects of human language processing. There are two main reasons for such development:

1. To develop automated tools for language processing
2. To gain a better understanding of human communication

Building computational models with human language-processing abilities requires a knowledge of how humans acquire, store, and process language. It also requires a knowledge of the world and of language.

Historically, there have been two major approaches to NLP—the *rationalist* approach and the *empiricist* approach. Early NLP research took a rationalist approach, which assumes the existence of some language faculty in the human brain. Supporters of this approach argue that it is not possible for children to learn a complex thing like natural language from limited sensory inputs. Empiricists do not believe in existence of a language faculty. Instead, they believe in the existence of some general organization principles such as pattern recognition, generalization, and association. Learning of detailed structures can, therefore, take place through the application of these principles on sensory inputs available to the child.

1.2 ORIGINS OF NLP

Natural language processing sometimes mistakenly termed natural language understanding—originated from machine translation research. While natural language understanding involves only the interpretation of language, natural language processing includes both understanding (interpretation) and generation (production). The NLP also includes speech processing. However, in this book, we are concerned with text processing only, covering work in the area of computational linguistics, and the tasks in which NLP has found useful application.

Computational linguistics is similar to theoretical- and psycho-linguistics, but uses different tools. Theoretical linguists mainly provide structural description of natural language and its semantics. They are not concerned with the actual processing of sentences or generation of sentences from structural description. They are in a quest for principles that remain common across languages and identify rules that capture linguistic generalization. For example, most languages have constructs like noun and verb phrases. Theoretical linguists identify rules that describe and restrict the structure of languages (grammar). Psycholinguists explain how humans produce and comprehend natural language. Unlike theoretical linguists, they are interested in the representation of linguistic structures as well as in the process by which these structures are produced. They rely primarily on empirical investigations to back up their theories.

Computational linguistics is concerned with the study of language using computational models of linguistic phenomena. It deals with the application of linguistic theories and computational techniques for NLP. In computational linguistics, representing a language is a major problem; most knowledge representations tackle only a small part of knowledge.

Representing the whole body of knowledge is almost impossible. The words knowledge and language should not be confused. This is discussed in detail in Section 1.3.

Computational models may be broadly classified under knowledge-driven and data-driven categories. Knowledge-driven systems rely on explicitly coded linguistic knowledge, often expressed as a set of handcrafted grammar rules. Acquiring and encoding such knowledge is difficult and is the main bottleneck in the development of such systems. They are, therefore, often constrained by the lack of sufficient coverage of domain knowledge. Data-driven approaches presume the existence of a large amount of data and usually employ some machine learning technique to learn syntactic patterns. The amount of human effort is less and the performance of these systems is dependent on the quantity of the data. These systems are usually adaptive to noisy data.

As mentioned earlier, this book is mainly concerned with computational linguistics approaches. We try to achieve a balance between semantic (knowledge-driven) and data-driven approaches on one hand, and between theory and practice on the other. It is at this point that the book differs significantly from other textbooks in this area. The tools and techniques have been covered to the extent that is needed to build sufficient understanding of the domain and to provide a base for application.

The NLP is no longer confined to classroom teaching and a few traditional applications. With the unprecedented amount of information now available on the web, NLP has become one of the leading techniques for processing and retrieving information. In order to cope with these developments, this book brings together information retrieval with NLP. The term *information retrieval* is used here in a broad manner to include a number of information processing applications such as information extraction, text summarization, question answering, and so forth. The distinction between these applications is made in terms of the level of detail or amount of information retrieved. We consider retrieval of information as part of processing. The word 'information' itself has a much broader sense. It includes multiple modes of information, including speech, images, and text. However, it is not possible to cover all these modes due to space constraints. Hence, this book focuses on textual information only.

1.3 LANGUAGE AND KNOWLEDGE

Language is the medium of expression in which knowledge is deciphered. We are not competent enough to define language and knowledge and its

implications. We are here considering the text from of the language and the content of it as knowledge.

Language, being a medium of expression, is the outer form of the content it expresses. The same content can be expressed in different languages. But can language be separated from its content? If so, how can the content itself be represented? Generally, the meaning of one language is written in the same language (but with a different set of words). It may also be written in some other, formal, language. Hence, to process a language means to process the content of it. As computers are not able to understand natural language, methods are developed to map its content in a formal language. Sometimes, formal language content may have to be expressed in a natural language as well. Thus, in this book, language is taken up as a knowledge representation tool that has historically represented the whole body of knowledge and that has been modified, maybe through generation of new words, to include new ideas and situations. The language and speech community, on the other hand, considers a language as a set of sounds that, through combinations, conveys meaning to a listener. However, we are concerned with representing and processing text only. Language (text) processing has different levels, each involving different types of knowledge. We now discuss various levels of processing and the types of knowledge it involves.

The simplest level of analysis is *lexical analysis*, which involves analysis of words. Words are the most fundamental unit (syntactic as well as semantic) of any natural language text. Word-level processing requires morphological knowledge, i.e., knowledge about the structure and formation of words from basic units (morphemes). The rules for forming words from morphemes are language specific.

The next level of analysis is *syntactic analysis*, which considers a sequence of words as a unit, usually a sentence, and finds its structure. Syntactic analysis decomposes a sentence into its constituents (or words) and identifies how they relate to each other. It captures grammaticality or non-grammaticality of sentences by looking at constraints like word order, number, and case agreement. This level of processing requires syntactic knowledge, i.e., knowledge about how words are combined to form larger units such as phrases and sentences, and what constraints are imposed on them. Not every sequence of words results in a sentence. For example, 'I went to the market' is a valid sentence whereas 'went the I market to' is not. Similarly, 'She is going to the market ' is valid, but 'She are going to the market' is not. Thus, this level of analysis requires detailed knowledge about rules of grammar.

Yet another level of analysis is *semantic analysis*. Semantics is associated with the meaning of the language. Semantic analysis is concerned with creating meaningful representation of linguistic inputs. The general idea of semantic interpretation is to take natural language sentences or utterances and map them onto some representation of meaning. Defining meaning components is difficult as grammatically valid sentences can be meaningless. One of the famous examples is, 'Colorless green ideas sleep furiously' (Chomsky 1957). The sentence is well-formed, i.e., syntactically correct, but semantically anomalous. However, this does not mean that syntax has no role to play in meaning. Bach (2002) considers:

> '... *semantics to be a projection of its syntax. That is semantic structure is interpreted syntactic structure.*'

But definitely, syntax is not the only component to contribute meaning. Our conception of meaning is quite broad. We feel that humans apply all sorts of knowledge (i.e., lexical, syntactic, semantic, discourse, pragmatic, and world knowledge) to arrive at the meaning of a sentence. The starting point in semantic analysis, however, has been lexical semantics (meaning of words). A word can have a number of possible meanings associated with it. But in a given context, only one of these meanings participates. Finding out the correct meaning of a particular use of word is necessary to find meaning of larger units. However, the meaning of a sentence cannot be composed solely on the basis of the meaning of its words. Consider the following sentences:

Kabir and Ayan are married. (1.1a)

Kabir and Suha are married. (1.1b)

Both sentences have identical structures, and the meanings of individual words are clear. But most of us end up with two different interpretations. We may interpret the second sentence to mean that Kabir and Suha are married to each other, but this interpretation does not occur for the first sentence. Syntactic structure and compositional semantics fail to explain these interpretations. We make use of pragmatic information. This means that semantic analysis requires pragmatic knowledge besides semantic and syntactic knowledge.

A still higher level of analysis is *discourse analysis*. Discourse-level processing attempts to interpret the structure and meaning of even larger units, e.g., at the paragraph and document level, in terms of words, phrases, clusters, and sentences. It requires the resolution of anaphoric references and identification of discourse structure. It also requires discourse knowledge, that is, knowledge of how the meaning of a sentence is determined by preceding sentences—e.g., how a pronoun refers to the

preceding noun—and how to determine the function of a sentence in the text. In fact, pragmatic knowledge may be needed for resolving anaphoric references. For example, in the following sentences, resolving the anaphoric reference 'they' requires pragmatic knowledge:

The district administration refused to give the trade union permission for the meeting because they feared violence. (1.2a)

The district administration refused to give the trade union permission for the meeting because they oppose government. (1.2b)

The highest level of processing is *pragmatic analysis*, which deals with the purposeful use of sentences in situations. It requires knowledge of the world, i.e., knowledge that extends beyond the contents of the text. The Cyc project (Lenat 1986) at University of Austin is an attempt to utilize world knowledge in NLP. However, its usefulness in a general-domain NLP system is yet to be demonstrated. Furthermore, whether or not semantics can be associated with a symbol manipulator and whether humans use logic in the same way as the Cyc project, are both issues of debate.

1.4 THE CHALLENGES OF NLP

There are a number of factors that make NLP difficult. These relate to the problems of representation and interpretation. Language computing requires precise representation of content. Given that natural languages are highly ambiguous and vague, achieving such representation can be difficult. The inability to capture all the required knowledge is another source of difficulty. It is almost impossible to embody all sources of knowledge that humans use to process language. Even if this were done, it is not possible to write procedures that imitate language processing as done by humans. In this section, we detail some of the problems associated with NLP.

Perhaps the greatest source of difficulty in natural language is identifying its semantics. The principle of compositional semantics considers the meaning of a sentence to be a composition of the meaning of words appearing in it. In the earlier section, we saw a number of examples where this principle failed to work. Our viewpoint is that words alone do not make a sentence. Instead, it is the words as well as their syntactic and semantic relation that give meaning to a sentence. As pointed out by Wittgenstein (1953): 'The meaning of a word is its use in the language.' A language keeps on evolving. New words are added continually and existing

words are introduced in new context. For example, most newspapers and TV channels use 9/11 to refer to the terrorist act on the World Trade Centre in the USA in 2004. When we process written text or spoken utterances, we have access to underlying mental representation. The only way a machine can learn the meaning of a specific word in a message is by considering its context, unless some explicitly coded general world or domain knowledge is available. The context of a word is defined by co-occurring words. It includes everything that occurs before or after a word. The frequency of a word being used in a particular sense also affects its meaning. The English word 'while' was initially used to mean 'a short interval of time'. But now it is more in use as a conjunction. None of the usages of 'while' discussed in this chapter correspond to this meaning.

Idioms, metaphor, and ellipses add more complexity to identify the meaning of the written text. As an example, consider the sentence:

The old man finally kicked the bucket. (1.3)

The meaning of this sentence has nothing to do with the words 'kick' and 'bucket' appearing in it.

Quantifier-scoping is another problem. The scope of quantifiers (the, each, etc.) is often not clear and poses problem in automatic processing.

The ambiguity of natural languages is another difficulty. These go unnoticed most of the times, yet are correctly interpreted. This is possible because we use explicit as well as implicit sources of knowledge. Communication via language involves two brains not just one—the brain of the speaker/writer and that of the hearer/reader. Anything that is assumed to be known to the receiver is not explicitly encoded. The receiver possesses the necessary knowledge and fills in the gaps while making an interpretation. As humans, we are aware of the context and current cultural knowledge, and also of the language and traditions, and utilize these to process the meaning. However, incorporating contextual and world knowledge poses the greatest difficulty in language computing. An example of cultural impact on language is the representation of different shades of white in the Eskimo world. It may be hard for a person living in plain to distinguish among various shades. Similarly, to an Indian, the word 'Taj' may mean a monument, a brand of tea, or a hotel, which may not be so for a non-Indian. Let us now take a look at the various sources of ambiguities in natural languages.

The first level of ambiguity arises at the word level. Without much effort, we can identify words that have multiple meanings associated with

them, e.g., bank, can, bat, and still. A word may be ambiguous in its part-of-speech or it may be ambiguous in its meaning. The word 'can' is ambiguous in its part-of-speech whereas the word 'bat' is ambiguous in its meaning. We hardly consider all possible meanings of a word to get the correct one. A program on the other hand, must be explicitly coded to resolve each meaning. Hence, we need to develop various models and algorithms to resolve them. Deciding whether 'can' is a noun or a verb is solved by 'part-of-speech tagging' whereas identifying whether a particular use of 'bank' corresponds to 'financial institution' sense or 'river bank' sense is solved by 'word sense disambiguation'. 'Part-of-speech tagging' and 'word sense disambiguation' algorithms are discussed in Chapters 3 and 5 respectively.

A sentence may be ambiguous even if the words are not, for example, the sentence: '*Stolen rifle found by tree.*' None of the words in this sentence is ambiguous but the sentence is. This is an example of structural ambiguity. Verb sub-categorization may help to resolve this type of ambiguity but not always. Probabilistic parsing, which is discussed in Chapter 4, is another solution. At a still higher level are pragmatic and discourse ambiguities. Ambiguities are discussed in Chapter 5.

A number of grammars have been proposed to describe the structure of sentences. However, there are an infinite number of ways to generate them, which makes writing grammar rules, and grammar itself, extremely complex. On top of it, we often make correct semantic interpretations of non-grammatical sentences. This fact makes it almost impossible for grammar to capture the structure of all and only meaningful text.

1.5 LANGUAGE AND GRAMMAR

Automatic processing of language requires the rules and exceptions of a language to be explained to the computer. Grammar defines language. It consists of a set of rules that allows us to parse and generate sentences in a language. Thus, it provides the means to specify natural language. These rules relate information to coding devices at the language level—not at the world-knowledge level (Bharati et al. 1995). However, since world knowledge affects both the coding (i.e., words) and the coding convention (structure), this blurs the boundary between syntax and semantics. Nevertheless such a separation is made because of the ease of processing and grammar writing.

The main hurdle in language specification comes from the constantly changing nature of natural languages and the presence of a large number

of hard-to-specify exceptions. Several efforts have been made to provide such specifications, which has led to the development of a number of grammars. Main among them are transformational grammar (Chomsky 1957), lexical functional grammar (Kaplan and Bresnan 1982), government and binding (Chomsky 1981), generalized phrase structure grammar, transformational grammar (Chomsky 1957), dependency grammar, Paninian grammar, and tree-adjoining grammar (Joshi 1985). Some of these grammars focus on derivation (e.g., phrase structure grammar) while others focus on relationships (e.g., dependency grammar, lexical functional grammar, Paninian grammar, and link grammar). We discuss some of these in Chapter 2. The greatest contribution to grammar comes from Noam Chomsky, who proposed a hierarchy of formal grammar based on level of complexity. These grammars use phrase structure rules (or rewrite rules). The term 'generative grammar' is often used to refer to the general framework introduced by Chomsky. Generative grammar basically refers to any grammar that uses a set of rules to specify or generate all and only grammatical (well-formed) sentences in a language. Chomsky argued that phrase structure grammars are not adequate to specify natural language. He proposed a complex system of transformational grammar in his book on *Syntactic Structures* (1957), in which he suggested that each sentence in a language has two levels of representation, namely, a deep structure and a surface structure (See Figure 1.1). The mapping from deep structure to surface structure is carried out by transformations. In the following paragraphs, we introduce transformational grammar.

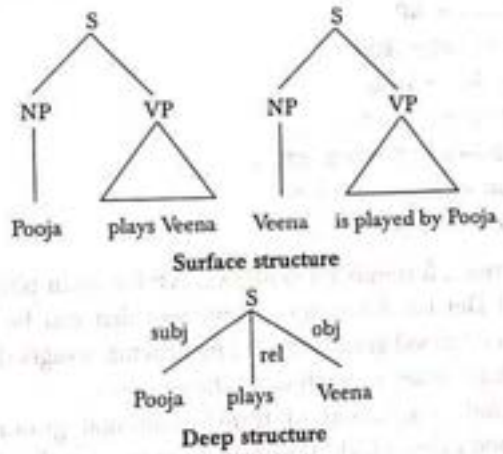


Figure 1.1 Surface and deep structures of sentence

Transformational grammar was introduced by Chomsky in 1957. Chomsky argued that an utterance is the surface representation of a 'deeper structure' representing its meaning. The deep structure can be transformed in a number of ways to yield many different surface-level representations. Sentences with different surface-level representations having the same meaning, share a common deep-level representation. Chomsky's theory was able to explain why sentences like

Pooja plays veena. (1.4a)

Veena is played by Pooja. (1.4b)

have the same meaning, despite having different surface structures (rules of subject and object are inverted). Both the sentences are being generated from the same 'deep structure' in which the deep subject is Pooja and the deep object is the veena.

Transformational grammar has three components:

1. Phrase structure grammar
2. Transformational rules
3. Morphophonemic rules—These rules match each sentence representation to a string of phonemes.

Each of these components consists of a set of rules. Phrase structure grammar consists of rules that generate natural language sentences and assign a structural description to them. As an example, consider the following set of rules:

S → NP + VP
VP → V + NP
NP → Det + Noun
V → Aux + Verb
Det → the, a, an, ...
Verb → catch, write, eat, ...
Noun → police, snatcher, ...
Aux → will, is, can, ...

In these rules, S stands for sentence, NP for noun phrase, VP for verb phrase, and Det for determiner. Sentences that can be generated using these rules are termed grammatical. The structure assigned by the grammar is a constituent structure analysis of the sentence.

The second component of transformational grammar is a set of transformation rules, which transform one phrase-maker (underlying) into another phrase-marker (derived). These rules are applied on the terminal

string generated by phrase structure rules. Unlike phrase structure rules, transformational rules are heterogeneous and may have more than one symbol on their left hand side. These rules are used to transform one surface representation into another, e.g., an active sentence into passive one. The rule relating active and passive sentences (as given by Chomsky) is

$$NP_1 - Aux - V - NP_2 \rightarrow NP_2 - Aux + be + en - V - by + NP_1$$

This rule says that an underlying input having the structure NP-Aux-V-NP can be transformed to NP – Aux + be + en – V – by + NP. This transformation involves addition of strings 'be' and 'en' and certain rearrangements of the constituents of a sentence. Transformational rules can be obligatory or optional. An obligatory transformation is one that ensures agreement in number of subject and verb, etc., whereas an optional transformation is one that modifies the structure of a sentence while preserving its meaning. Morphophonemic rules match each sentence representation to a string of phonemes.

Consider the active sentence:

The police will catch the snatcher. (1.5)

The application of phrase structure rules will assign the structure shown in Figure 1.2 to this sentence.

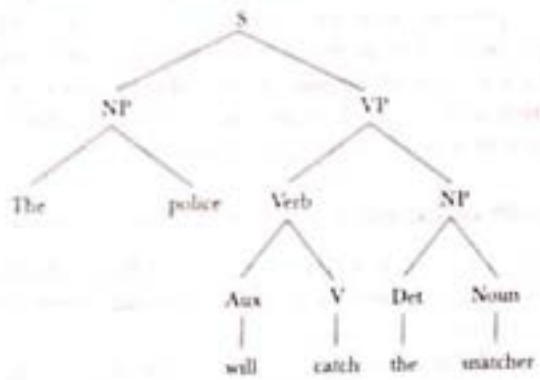


Figure 1.2 Parse structure of sentence (1.5)

The passive transformation rules will convert the sentence into:
The + culprit + will + be + en + catch + by + police (Figure 1.3).

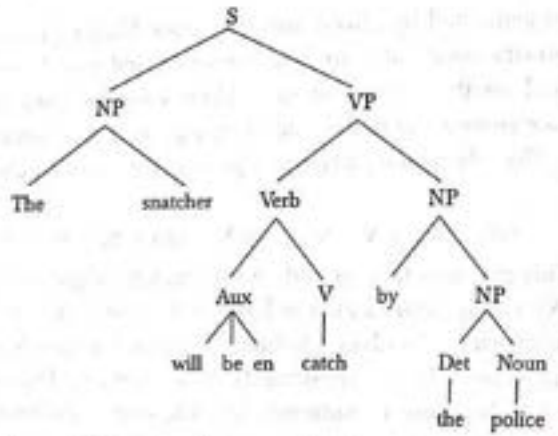


Figure 1.3 Structure of sentence (1.5) after applying passive transformations

Another transformational rule will then reorder 'en + catch' to 'catch + en' and subsequently one of the morphophonemic rules will convert 'catch + en' to 'caught'. In general, the noun phrase is not always as simple as in sentence (1.5). It may contain other embedded structures, such as adjectives, modifiers, relative clause, etc. Long distance dependencies are other language phenomena that cannot be adequately handled by phrase structure rules. Long distance dependency refers to syntactic phenomena where a verb and its subject or object can be arbitrarily apart. The problem in the specification of appropriate phrase structure rules occurs because these phenomena cannot be localized at the surface structure level (Joshi and Vijayshanker 1989). Wh-movement[1] are a specific case of these types of dependencies.

1.6 PROCESSING INDIAN LANGUAGES

There are a number of differences between Indian languages and English. This introduces differences in their processing. Some of these differences are listed here.

- Unlike English, Indic scripts have a non-linear structure.
- Unlike English, Indian languages have SOV (Subject-Object-Verb) as the default sentence structure.

[1]It refers to a syntactic phenomenon in which interrogative words, called wh-words, appear at the beginning of the sentence. For example, when the direct object of the verb 'read' in the sentence 'She is reading a book' is replaced with a wh-word, the sentence becomes 'What is she reading?' instead of 'She is reading what?'.

- Indian languages have a free word order, i.e., words can be moved freely within a sentence without changing the meaning of the sentence.
- Spelling standardization is more subtle in Hindi than in English.
- Indian languages have a relatively rich set of morphological variants.
- Indian languages make extensive and productive use of complex predicates (CPs).
- Indian languages use post-position (*Karakas*) case markers instead of prepositions.
- Indian languages use verb complexes consisting of sequences of verbs, e.g., गा रहा है (*ga raha hai*—singing) and खेल रही है (*khel rahi hai*—playing). The auxiliary verbs in this sequence provide information about tense, aspect, modality, etc.

Except for the direction in which its script is written, Urdu is closely related to Hindi. Both share similar phonology, morphology, and syntax. Both are free-word-order languages and use post-positions. They also share a large amount of their vocabulary. Differences in the vocabulary arise mainly because a significant portion of Urdu vocabulary comes from Persian and Arabic, while Hindi borrows much of its vocabulary from Sanskrit.

Paninian grammar provides a framework for Indian language models. These can be used for computation of Indian languages. The grammar focuses on extraction of Karaka relations from a sentence. We talk about the details of modelling in Chapter 2. A parsing framework based on Paninian grammar is introduced in Chapter 4 and issues involved in Indian language translation (using Paninian grammar theory) are discussed in Chapter 8.

1.7 NLP APPLICATIONS

Machine translation is the first application area of NLP. It involves the complete linguistic analysis of a natural language sentence, and linguistic generation of an output sentence. It is one of the most comprehensive and most challenging tasks in the area (AI-complete). However, the recent dramatic progress in the field of NLP has found interesting applications in information retrieval, information extraction, text summarization, etc. This book offers an extensive coverage of these recent applications, and also of traditional ones like machine translation and natural language generation. The focus has been on bridging the gap between theory and practice rather than on offering a gamut of linguistic, psychological, and computational theories.

The applications utilizing NLP include the following.

Machine Translation

This refers to automatic translation of text from one human language to another. In order to carry out this translation, it is necessary to have an understanding of words and phrases, grammars of the two languages involved, semantics of the languages, and world knowledge.

Speech Recognition

This is the process of mapping acoustic speech signals to a set of words. The difficulties arise due to wide variations in the pronunciation of words, homonym (e.g. dear and deer) and acoustic ambiguities (e.g., in the rest and interest).

Speech Synthesis

Speech synthesis refers to automatic production of speech (utterance of natural language sentences). Such systems can read out your mails on telephone, or even read out a storybook for you. In order to generate utterances, text has to be processed. So, NLP remains an important component of any speech synthesis system.

Natural Language Interfaces to Databases

Natural language interfaces allow querying a structured database using natural language sentences.

Information Retrieval

This is concerned with identifying documents relevant to a user's query. NLP techniques have found useful applications in information retrieval. Indexing (stop word elimination, stemming, phrase extraction, etc.), word sense disambiguation, query modification, and knowledge bases have also been used in IR system to enhance performance, e.g., by providing methods for query expansion. WordNet, LDOCE (*Longman Dictionary of Contemporary English*) and *Roget's Thesaurus* are some of the useful lexical resources for IR research.

Information Extraction

An information extraction system captures and outputs factual information contained within a document. Similar to an information retrieval system, it responds to a user's information need. However, unlike in an information retrieval system, the information need is not expressed as a keyword query. Instead it is specified as pre-defined database schemas or templates. Whereas an IR system identifies a subset of documents in a large repository of text database—e.g., in a library scenario, a subset of resources in a library—an information extraction system identifies a subset of information within a document that fits the pre-defined template.

Question Answering

Given a question and a set of documents, a question answering system attempts to find the precise answer, or at least the precise portion of text in which the answer appears. This is unlike an IR system, which returns whole document that seems relevant to the user's query. A question answering system is different from an information extraction system in that the content that is to be extracted is unknown. In general, a question answering system benefits from having an information extraction system to identify entities in the text. A question answering system requires more NLP than an information retrieval system or an information extraction system. It requires not only precise analysis of questions and portions of texts but also semantic as well as background knowledge to answer certain type of questions.

Text Summarization

This deals with the creation of summaries of documents and involves syntactic, semantic, and discourse level processing of text.

1.8 SOME SUCCESSFUL EARLY NLP SYSTEMS

***ELIZA* (Weizenbaum 1966)**

ELIZA is one of the earliest natural language understanding programs. It uses syntactic patterns to mimic human conversation with the user. Here is a sample conversation.

Eliza: Hello. I am ELIZA. How may I help you?
User: I am feeling a little bit sleepy.
Eliza: How long have you been feeling a little bit sleepy?
User: For almost half an hour.
Eliza: Please go on.

***SysTran* (System Translation)**

The first SysTran machine translation system was developed in 1969 for Russian-English translation. SysTran also provided the first on-line machine translation service called Babel Fish, which is used by AltaVista search engines for handling translation requests from users.

TAUM METEO

This is a natural language generation system used in Canada to generate weather reports. It accepts daily weather data and generates weather reports in English and French.

SHRDLU (Winogard 1972)

This is a natural language understanding system that simulates actions of a robot in a block world domain. It uses syntactic parsing and semantic reasoning to understand instructions. The user can ask the robot to manipulate the blocks, to tell the blocks configurations, and to explain its reasoning.

LUNAR (Woods 1977)

This was an early question answering system that answered questions about moon rock.

1.9 INFORMATION RETRIEVAL

The availability of a large amount of text in electronic form has made it extremely difficult to get relevant information. Information retrieval systems aim at providing a solution to this.

The term 'information' should not be confused with the term 'entropy' (numerical measure of the uncertainty of an outcome) as it is used in communication theory. Information is being used here to reflect 'subject matter' or the 'content' of some text. We are not interested in 'digital communication', where bits and bytes are the information carriers. Instead our focus in on the communication taking place between human beings as expressed through natural languages. Information is always associated with some data (text, number, image, and so on): we are concerned with text only. Hence, we consider words as the carriers of information and written text as the message encoded in natural language.

As a cognitive activity, the word 'retrieval' refers to operation of accessing information from memory. We use the word 'retrieval' to refer to the operation of accessing information from some computer-based representation. Retrieval of information thus requires information to be processed and stored. Not all the information represented in computable form is retrieved. Instead, only the information relevant to the needs expressed in the form of query is located. In order to get this relevance, the stored and processed information needs to be compared against query representation. Information retrieval (IR) deals with all these facets. It is concerned with the organization, storage, retrieval, and evaluation of information relevant to the query.

Information retrieval deals with unstructured data. The retrieval is performed based on the content of the document rather than on its structure. The IR systems usually return a ranked list of documents. The IR components have been traditionally incorporated into different types

of information systems including database management systems, bibliographic text retrieval systems, question answering systems, and more recently in search engines.

Current approaches for accessing large text collections can be broadly classified into two categories. The first category consists of approaches that construct topic hierarchy, e.g., Yahoo. This helps the user locate documents of interest manually by traversing the hierarchy. However, it requires manual classification of new documents within the existing taxonomy. This makes it cost ineffective and inapplicable due to rapid growth of documents on the Web. The second category consists of approaches that rank the retrieved documents according to relevance. We discuss various IR models that support ranked retrieval in Chapter 9.

Major Issues in Information Retrieval (Siddiqui 2006)

There are a number of issues involved in the design and evaluation of IR systems, which are briefly discussed in this section. The first important point is to choose a representation of the document. Most human knowledge is coded in natural language, which is difficult to use as knowledge representation language for computer systems. Most of the current retrieval models are based on keyword representation. This representation creates problems during retrieval due to polysemy, homonymy, and synonymy. Polysemy involves the phenomenon of a lexeme with multiple meaning. Homonymy is an ambiguity in which words that appear the same have unrelated meanings. Ambiguity makes it difficult for a computer to automatically determine the conceptual content of documents. Synonymy creates problem when a document is indexed with one term and the query contains a different term, and the two terms share a common meaning. Another problem associated with keyword-based retrieval is that it ignores semantic and contextual information in the retrieval process. This information is lost in the extraction of keywords from the text and cannot be recovered by the retrieval algorithms.

A related issue is that of inappropriate characterization of queries by the user. There can be many reasons for the vagueness and inaccuracy of the user's queries, say for instance, her lack of knowledge of the subject or even the inherent vagueness of the natural language. The user may fail to include relevant terms in the query or may include irrelevant terms. Inappropriate or inaccurate queries lead to poor retrieval performance. The problem of ill-specified query can be dealt with by modifying or expanding queries. An effective technique based on user-interaction is relevance feedback which modifies queries based on the feedback provided by the user on initial retrieval.

In order to satisfy the user's request, an IR system matches document representation with query representation. Matching query representation with that of the document is another issue. A number of measures have been proposed to quantify the similarity between a query and the document to produce a ranked list of results. Selection of the appropriate similarity measure is a crucial issue in the design of IR systems.

Evaluating the performance of IR systems is also a major issue. There are many aspects of evaluation, the most important being the effectiveness of the system. Recall and precision are the most widely used measures of effectiveness.

As the major goal of IR is to search a document in a manner relevant to the query, understanding what constitutes relevance is also an important issue. Relevance is subjective in nature (Saracevic 1991). Only the user can tell the true relevance; it is not possible to measure this 'true relevance'. One may however, define the degree of relevance. Relevance has been considered as a binary concept, whereas it is in fact a continuous function (a document may be exactly what the user wants or it may be closely related). Current evaluation techniques do not support this continuity as it is quite difficult to put into practice. A number of relevance frameworks have been proposed (Saracevic 1996). These include the system, communication, psychological, and situational frameworks. The most inclusive is the situational framework, which is based on the cognitive view of the information seeking process and considers the importance of situation, context, multi-dimensionality, and time. A survey of relevance studies can be found in Mizzaro (1997). Most of the evaluations of IR systems have so far been done on document test collections with known relevance judgments.

The size of document collections and the varying needs of users also complicate text retrieval. Some users require answers of limited scope, while others require documents with a wider scope. These differing needs can require different and specialized retrieval methods. However, these are research issues and have not been dealt with in this book.

SUMMARY

- Language is the primary means of communication used by humans.
- Natural language processing is concerned with the development of computational models of aspects of human language processing.
- Theoretical linguists are mainly interested in providing a description of the structure and semantics of natural language, whereas computational linguists deal with the study of language from a computational point of view.
- Historically, there have been two major approaches to natural language processing, namely rationalist approach and empiricist approach.
- The highly ambiguous and vague nature of natural language makes it difficult to create a representation amenable to computing.

REFERENCES

Bach, Kent, 2002, *Meaning and Truth*, J. Keim Campbell, M. O'Rourke, and D. Shei (Eds.), Seven Bridges Press, New York, pp. 284–92.

Chomsky, Noam, 1957, *Syntactic Structures*, Mouton, The Hague.

_____1981, *Lectures on Government and Binding*, Foris Publications, Dordrecht, The Netherlands.

Joshi, Aravind K., 1985, 'Tree adjoining grammar: How much sensitivity is required to provide reasonable structural description,' *Natural Language Parsing*, D. Dowty, L. Karttunen, and A. Zwicky (Eds.), Cambridge University Press, Cambridge.

Joshi, Aravind K. and K. Vijayshanker, 1989, 'Treatment of long distance dependencies in LFG and TAG: functional uncertainty in LFG is a corollary in TAG,' *Proceedings of the 27th Annual Meeting on Association for Computational Linguistics*, Vancouver, British Columbia, pp. 220–27.

Kaplan, R.M. and Joan Bresnan, 1982, 'Lexical functional grammar: A formal system for grammatical representation,' *The Mental Representation of Grammatical Relations*, Joan Bresnan (Ed.), MIT Press, Cambridge.

Lenat, D.B., M. Prakash, and M. Shepherd, 1986, 'Cyc: using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks,' *AI Magazine*, 6(4).

Mizzaro, S., 1997, 'Relevance: the whole history,' *Journal of the American Society for Information Science*, 48(9), pp. 810–32.

Saracevic, T., 1991, 'Individual differences in organizing, searching and retrieving information,' *Proceedings of the 54th Annual Meeting of the American Society for Information Science (ASIS)*, pp. 82–86.

_____1996, 'Relevance reconsidered,' *Proceedings of CoLIS 2, Second International Conference on Conceptions of Library and Information Science: Integration in Perspective*, P. Ingwersen and N.O. Pors (Eds.), The Royal School of Librarianship, Copenhagen, pp. 201–18.

Siddiqui, Tanveer, 2006, 'Intelligent techniques for effective information retrieval: a conceptual graph-based approach,' *Ph.D. Thesis*, J.K. Institute of Applied Physics, Deptt. of Electronics and Communication, University of Allahabad.

Weizenbaum, R., 1966, 'ELIZA—A computer program for the study of natural language communication between man and machine,' *Communications of the ACM*, 9(1).

Winogard, Terry, 1972, *Understanding Natural Language*, Academic Press, New York.

Woods, William, 1977, 'Lunar Rocks in Natural English: Explorations in Natural Language Question-answering,' *Linguistic Structures Processing*, A. Zampolli (Ed.), Elsevier, North Holland.

EXERCISES

1. Differentiate between the rationalist and empiricist approaches to natural language processing.
2. List the motivation behind the development of computational models of languages.
3. Briefly discuss the meaning components of a language.
4. What makes natural language processing difficult?
5. What is the role of transformational rules in transformational grammar? Explain with the help of examples.

CHAPTER 2

LANGUAGE MODELLING

CHAPTER OVERVIEW

The domain of language is quite vast. It presents an almost infinite number of sentences to the reader (or computer). To handle such a large number of sentences, we have to create a model of the domain, which can subsequently be simplified and handled computationally. A number of language models have been proposed. We introduce some of these models in this chapter. To create a general model of any language is a difficult task. There are two approaches for language modelling. One is to define a grammar that can handle the language. The other is to capture the patterns in a grammar language statistically. This chapter has a mixed approach. It gives a glimpse of both grammar-based model and statistical language model. These include lexical functional grammar, government and binding, Paninian grammar, and *n*-gram based model.

2.1 INTRODUCTION

Why and how do we model a language? This question has been discussed by linguists since 500 BC. Computational linguists also have to confront this question. It is obvious that our purpose is to understand and generate natural languages from a computational viewpoint. One approach can be to just take a language, try to understand every word and sentence of it, and then come to a conclusion. This approach has not succeeded as there are difficulties at each stage, which we will understand as we go through this book. An alternative approach is to study the grammar of various languages, compare them, and if possible, arrive at reasonable models that facilitate our understanding of the problem and designing of natural-language tools.

A model is a description of some complex entity or process. A language model is thus a description of language. Indeed, natural language is a complex entity and in order to process it through a computer-based program, we need to build a representation (model) of it. This is known

as *language modelling*. Language modelling can be viewed either as a problem of grammar inference or a problem of probability estimation. A grammar-based language model attempts to distinguish a grammatical sentence from a non-grammatical (ill-formed) one, whereas a probabilistic language model attempts to identify a sentence based on a probability measure, usually a maximum likelihood estimate. These two viewpoints have led to the following categorization of language modelling approaches.

Grammar-based language model

A grammar-based approach uses the grammar of a language to create its model. It attempts to represent the syntactic structure of language. Grammar consists of hand-coded rules defining the structure and ordering of various constituents appearing in a linguistic unit (phrase, sentence, etc.). For example, a sentence usually consists of noun phrase and a verb phrase. The grammar-based approach attempts to utilize this structure and also the relationships between these structures.

Statistical language modelling

The statistical approach creates a language model by training it from a corpus. In order to capture regularities of a language, the training corpus needs to be sufficiently large. Rosenfeld (1994) pointed out:

> *Statistical language modelling (SLM) is the attempt to capture regularities of natural language for the purpose of improving the performance of various natural language applications.*

Statistical language modelling is one of the fundamental tasks in many NLP applications, including speech recognition, spelling correction, handwriting recognition, and machine translation. It has now found applications in information retrieval, text summarization, and question answering also. A number of statistical language models have been proposed in literature. The most popular of these are the *n*-gram models. We discuss this model in Section 2.3. The following section discusses various grammar-based models.

2 VARIOUS GRAMMAR-BASED LANGUAGE MODELS

Various computational grammars have been proposed and studied, e.g., transformational grammar (Chomsky 1957), lexical functional grammar (Kaplan and Bresnan 1982), government and binding (Chomsky 1981), generalized phrase structure grammar (Gazdar et al. 1985), dependency grammar, paninian grammar, and tree-adjoining grammar (Joshi 1985).

This section focuses on lexical functional grammar (LFG), generalized phrase structure grammar (GPSG), government and binding (GB), and Paninian grammar (PG) and introduces various approaches to understand a language in a grammatical and rule-based format. It also introduces the dominant approaches to create statistical models of language and grammar.

2.2.1 Generative Grammars

In 1957, in his book on *Syntactic Structures*, Noam Chomsky wrote that we can generate sentences in a language if we know a collection of words and rules in that language. Only those sentences that can be generated as per the rules are grammatical. This point of view has dominated computational linguistics and is appropriately termed generative grammar. The same idea can be used to model a language. If we have a complete set of rules that can generate all possible sentences in a language, those rules provide a model of that language. Of course, we are talking only about the syntactical structure of language here.

Language is a relation between the sound (or the written text) and its meaning. Thus, any model of a language should also deal with the meaning of its sentences. As seen earlier, we can have a perfectly grammatical but meaningless sentence.

In this chapter, we will assume that grammars are a type of language models.

2.2.2 Hierarchical Grammar

Chomsky (1956) described classes of grammars in a hierarchical manner, where the top layer contained the grammars represented by its sub classes. Hence, Type 0 (or unrestricted) grammar contains Type 1 (or context-sensitive grammar), which in turn contains Type 2 (context-free grammar) and that again contains Type 3 grammar (regular grammar). Although this relationship has been given for classes of formal grammars, it can be extended to describe grammars at various levels, such as in a class-sub class (embedded) relationship.

2.2.3 Government and Binding (GB)

As discussed in Chapter 1, a common viewpoint taken by linguists (not computational linguists, however) is that the structure of a language (or how well its sentences are formed) can be understood at the level of its meaning, particularly while resolving structural ambiguity. However, the sentences are given at the syntactical level and the transformation from meaning to syntax or vice versa is not well understood.

Transformational grammars assume two levels of existence of sentences—one at the surface level and the other at the deep root level (this should not be confused with the meaning level). Government and binding (GB) theories have renamed them as s-level and d-level, and identified two more levels of representation (parallel to each other) called *phonetic form* and *logical form*. According to GB theories, language can be considered for analysis at the levels shown in Figure 2.1.

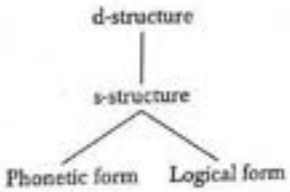


Figure 2.1 Different levels of representation in GB

If we describe language as the representation of some 'meaning' in a 'sound' form, then according to Figure 2.1, these two ends are the logical form (LF) and phonetic form (PF) respectively. The GB is concerned with LF, rather than PF. Chomsky was the first to put forward a GB theory (Peter Sells 1985).

Transformational grammars have hundreds of rewriting rules, which are generally language-specific and also construct-specific (say, different rules for assertive and interrogative sentences in English, or for active and passive voice sentences). Generation of a complete set of coherent rules may not be possible. The GB envisages that if we define rules for structural units at the deep level, it will be possible to generate any language with fewer rules. These deep-level structures are abstractions of noun-phrase, verb-phrase, etc., and common to all languages. It is possible to do if, as GB theory states, a child learns its mother tongue because the human mind is 'hard-wired' with some universal grammar rules. Thus, the data enters the mind and its abstract structure gives rise to actual phonetic structures. The existence of deep level, language-independent, abstract structures, and the expression of these in surface level, language-specific structures with the help of simple rules is the main concern of GB theories. Let us take an example to explain d- and s-structures.

Example 2.1

Mukesh was killed. (2.1)

(i) In transformational grammar, this can be represented as S-NP Aux VP as given below:

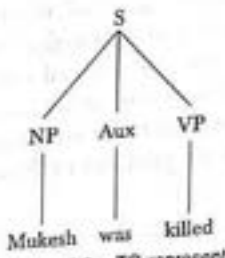


Figure 2.2 TG representation of sentence (2.1)

(ii) In GB, the s-structure and d-structure are as follows:

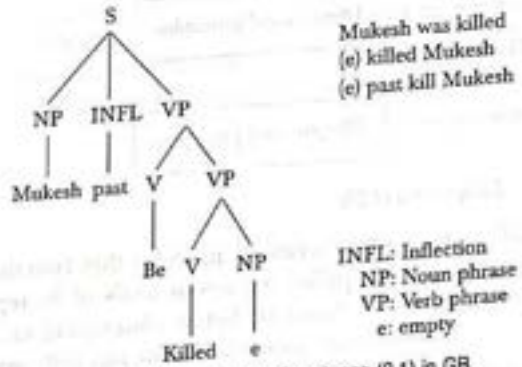


Figure 2.3 Surface structure of sentence (2.1) in GB

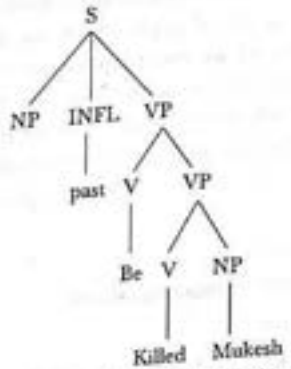


Figure 2.4 Deep structure of sentence (2.1) in GB

Components of GB

Government and binding (GB) comprises a set of theories that map the structures from d-structure to s-structure and to logical form (LF) (leaving aside the phonetic form). A general transformational rule called 'Move α' is applied at d-structure level as well as at s-structure level. This can move constituents at any place if it does not violate the constraints put by several theories and principles. Hence, in its simplest from GB can be represented by Figure 2.5.

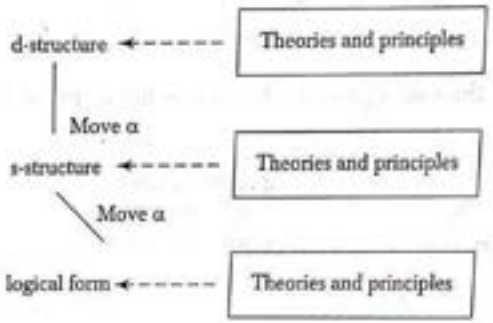


Figure 2.5 Components of GB

Hence, GB consists of 'a series of modules that contain constraints and principles' (Sells 1985) applied at various levels of its representations and the transformation rule, Move α. Before elaborating on these modules—which include X-bar theory, projection principle, θ-theory and θ-criterion, C-command and government, case theory, empty category principle (ECP), and binding theory—we discuss the general characteristics of GB.

The GB considers all three levels of representations (d-, s-, and LF) as syntactic, and LF is also related to meaning or semantic-interpretive mechanisms. However, GB applies the same Move α transformation to map d-levels to s-levels or s-levels to LF level. LF level helps in quantifier scoping and also in handling various sentence constructions such as passive or interrogative constructions. An example of LF representation may be helpful.

Example 2.2 Consider the sentence:

Two countries are visited by most travellers. (2.2)

Its two possible logical forms are:

LF1: [$_s$ Two countries are visited by [$_{NP}$ most travellers]]

LF2: Applying Move α

[$_{NP}$ Most travellers$_i$] [$_s$ two countries are visited by e$_i$]

In LF1, the interpretation is that most travellers visit the same two countries (say, India and China). In LF2, when we move [most travellers] outside the scope of the sentence, the interpretation can be that most travellers visit two countries, which may be different for different travellers.

One of the important concepts in GB is that of constraints. It is the part of the grammar which prohibits certain combinations and movements; otherwise Move α can move anything to any possible position. Thus, GB is basically the formulation of theories or principles which create constraints to disallow the construction of ill-formed sentences. To account for cross-lingual constraints of similar type, GB can specify that 'a constituent cannot be moved from position X' (where X can have value X_1 in one language, X_2 in another, and so on). These rules are so general and language-independent that 'language-particular details of description typically go uncharted in GB' (Sells 1985).

Figure 2.5 showed the application of various theories and principles at three different levels of representations in GB. Figure 2.6 mentions these explicitly to understand the organization of GB.

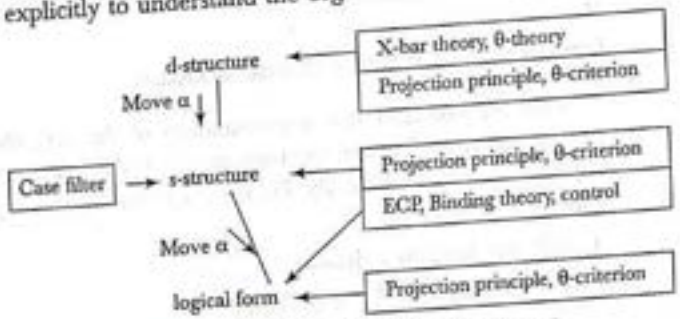


Figure 2.6 Organization of GB (adapted from Peter Sells 1985)

$\bar{X}$ Theory

The $\bar{X}$ theory (pronounced X-bar theory) is one of the central concepts in GB. Instead of defining several phrase structures and the sentence structure with separate sets of rules, $\bar{X}$ theory defines them both as maximal projections of some head. In this manner, the entities defined become language independent. Thus, noun phrase (NP), verb phrase (VP), adjective phrase (AP), and prepositional phrase (PP) are maximal projections of noun (N), verb (V), adjective (A), and preposition (P) respectively, and can be represented as head X of their corresponding phrases (where X = {N, V, A, P}). Not only that, even the sentence

structure (S′, which is projection of sentence) can be regarded as the maximal projection of inflection (INFL). The GB envisages projections at two levels—first the projection of head at semi-phrasal level, denoted by $\bar{X}$, and then the second maximal projection at the phrasal level, denoted by $\bar{\bar{X}}$.

For sentences, the first level projection is denoted as S and the second level maximal projection is denoted by S′. We now illustrate phrase and sentence representations with the help of examples.

Example 2.3 Figure 2.7 depicts the general and particular structures with examples. We see the general structure in Figure 2.7(a).

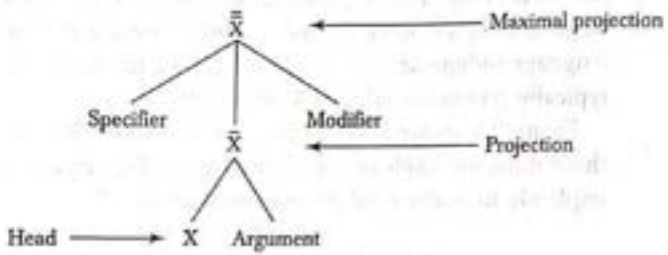


Figure 2.7(a) General phrase and sentential structure

Next, we consider the representation of the NP, *the food in a dhaba*. This is followed by the representation of VP, AP, and PP structure in Figure 2.7(c–e); and finally Figure 2.7(f) shows the representation of a sentence.

1. NP: the food in a dhaba

$[_{NP}\ the [_{N}\ food]_{PP}\ [in\ a\ dhabha]]$

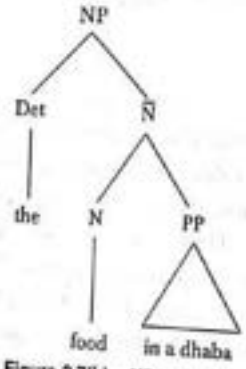


Figure 2.7(b) NP structure

2. VP: ate the food in a dhaba

$[_{VP}\ [_{V'}[_{V}\ ate][_{NP}\ the\ food]][_{PP}\ in\ a\ dhaba]]$

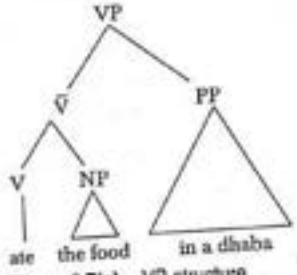


Figure 2.7(c) VP structure

3. AP: very proud of his country

$[_{AP}\ [_{Deg}\ very][_{\bar{A}}\ [_{A}\ proud][_{PP}\ of\ his\ country]]]$

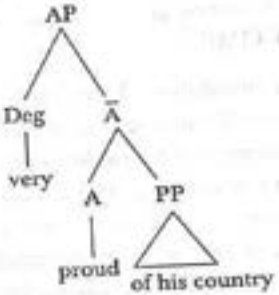


Figure 2.7(d) AP structure

4. PP: in a dhaba

$[_{PP}\ [_{\bar{P}}[_{P}\ in][_{NP}[_{Det}\ a][_{N}\ dhabha]]]]$

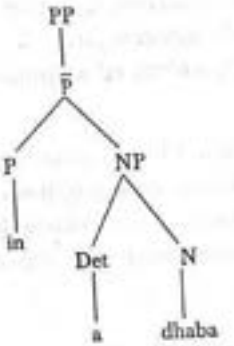


Figure 2.7(e) PP structure

5. S: that she ate the food in a dhaba

$[_{\bar{S}}[_{COMP}\ that][_{S}[_{Det}\ she][_{INFL}\ past][_{VP}\ ate\ the\ food\ in\ a\ dhaba]]]$

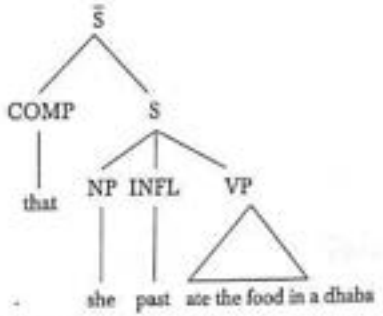


Figure 2.7(f) Maximal projection of sentence structure

As shown in Figure 2.7(f), the sentence is considered to be the head of INFL and the projection of sentence is denoted by $\bar{S}$, which has the specifier as complementizer (COMP).

Sub-categorization It is to be noted that GB does not consider traditional phrase structure as an appropriate device for defining language constructs. It places the burden of ascertaining well-formedness to sub-categorization frames of heads. In principle, any maximal projection can be the argument of a head, but sub-categorization is used as a filter to permit various heads to select a certain subset of the range of maximal projections. For example, we know that the verb 'eat' can sub-categorize for NP, whereas the verb 'sleep' cannot. Hence, 'ate food' is well-formed, but the sentence '*slept the bed' is not. GB claims that defining phrase structures as head projections and sub-categorization helps ensure well-formed structures, even at the sentence level.

As 'verb' is not the head of the sentence, it cannot sub-categorize for 'subject NP', while it can perfectly sub-categorize for 'object NP'. This explains, to certain extent, the 'subject/object' asymmetry (Sells 1985).

Projection Principle

The projection principle, a basic notion in GB, places a constraint on the three syntactic representations and their mapping from one to the other. The principle states that representations at all syntactic levels (i.e., d-level, s-level, and LF level) are projections from the lexicon. Thus, lexical

*will precede the incorrect sentences.

properties of categorical structure (sub-categorization) must be observed at each level.

This can be understood with an example. Suppose 'the object' is not present at d-level, then another NP cannot take this position at s-level. This principle, in conjunction with the possibility of presence of empty category and other theories (like Binding Theory) ensures correct movement and well-formed structure.

Theta Theory (θ-Theory) or The Theory of Thematic Relations

As discussed earlier, 'sub-categorization' only places a restriction on syntactic categories which a head can accept. GB puts another restriction on the lexical heads through which it assigns certain roles to its arguments. These roles are pre-assigned and cannot be violated at any syntactical level as per the projection principle. These role assignments are called theta-roles and are related to 'semantic-selection'.

Theta-role and Theta-criterion There are certain thematic roles from which a head can select. These are called θ-roles and they are mentioned in the lexicon, say for example the verb 'eat' can take arguments with θ-roles '(Agent, Theme)'. Agent is a special type of role which can be assigned by a head to outside arguments (external arguments) whereas other roles are assigned within its domain (internal arguments). Hence in 'Mukesh ate food', the verb 'eat' assigns the 'Agent' role to 'Mukesh' (outside VP) and 'Theme' (or 'patient') role to 'food'. As roles are assigned based on the syntactic positions of the arguments, it is important that there should be a match between the number of roles and number of arguments as depicted by θ-criterion.

Theta-Criterion states that 'each argument bears one and only one θ-role, and each θ-role is assigned to one and only one argument' (Sells 1985). Thus, each argument will have a unique θ-role and cannot be moved to a position where it may acquire another θ-role.

In GB, d-structure is conceived as some kind of 'pure' representation of arguments and hence, θ-roles are assigned at d-level only, whereas theta-criterion is applied at all the three levels, as shown in Figure 2.6.

C-command and Government

As 'Government' is a special case of 'C-command', we will first define C-command.

C-command C-command defines the scope of maximal projection. It is a basic mechanism through which many constraints are defined on Move

α. If any word or phrase (say α or β) falls within the scope of and is determined by a maximal projection, we say that it is dominated by the maximal projection. Now, if there are two structures α and β related in such a way that 'every maximal projection dominating α dominates β', we say that α C-commands β, and this is the necessary and sufficient condition (iff) for C-command.

The definition of C-command does not include all maximal projections dominating β, only those dominating α. If we put this extra constraint, it becomes a kind of mutual C-command (Sells 1985), called government.

Government

α governs β iff:
α C-commands β

α is an X (head, e.g., noun, verb, preposition, adjective, and inflection), and every maximal projection dominating β dominates α.

Thus no maximal projection can intervene between the governor and governee. In GB literature, this has been stated as: 'Maximal projections are barriers to government.'

Movement, Empty Category, and Co-indexing

Briefly let us discuss Move α. In GB, Move α is described as 'move anything anywhere', though it provides restrictions for valid movements.

In GB, the active to passive transformation is the result of NP movement as shown in sentence (2.3). Another well-known movement is the wh-movement, where wh-phrase is moved as follows.

What did Mukesh eat ? (2.3)
[Mukesh INFL eat what]

As discussed in the projection principle, lexical categories must exist at all the three levels. This principle, when applied to some cases of movement leads to the existence of an abstract entity called empty category. In GB, there are four types of empty categories, two being empty NP positions called wh-trace and NP trace, and the remaining two being pronouns called small 'pro' and big 'PRO'. This division is based on two properties—anaphoric (+a or −a) and pronominal (+p or −p).

Wh-trace	−a, −p
NP-trace	+a, −p
small 'pro'	−a, +p
big 'PRO'	+a, +p

Co-indexing is the indexing of the subject NP and AGR (agreement) at d-structure which are preserved by Move α operations at s-structure.

When an NP-movement takes place, a trace of the movement is created by having an indexed empty category (e_i) from the position at which the movement began to the corresponding indexed NP, i.e. NP_i. All A-positions (argument positions) at s-level are also freely indexed. These categories and indices are used to define Binding Theory.

It is interesting to note that for defining constraints to movement, the theory identifies two positions in a sentence. Positions assigned θ-roles are called θ-positions, while others are called $\bar{\theta}$-positions.

In a similar way, core grammatical positions (where subject, object, indirect object, etc., are positioned) are called A-positions (arguments positions), and the rest are called $\bar{A}$-positions.

Binding Theory

Binding is defined by Sells (1985) as follows:

α binds β iff
α C-commands β, and
α and β are co-indexed

As we noticed in sentence (2.1),

[e_i INFL kill Mukesh]
[$Mukesh_i$ was killed (by e_i)]
Mukesh was killed.

Empty clause (e_i) and Mukesh (NP_i) are bound. This theory gives a relationship between NPs (including pronouns and reflexive pronouns).

Now, binding theory can be given as follows:

(a) An anaphor (+a) is bound in its governing category.
(b) A pronominal (+p) is free in its governing category.
(c) An R-expression (−a, −p) is free.

This theory applies to binding at A-positions. Governing category is the local domain (the smallest only) NP or S containing it (G or p or R-expression) and its governor.

Example 2.4

A: $Mukesh_i$ knows $himself_i$
B: $Mukesh_i$ believes that Amrita knows him_i
C: Mukesh believes that $Amrita_j$ knows $Nupur_k$

Similar rules apply on empty categories also:

NP-trace: +a, −p: $Mukesh_i$ was killed e_i
wh-trace: −a, −p: Who_i does he_i like e_i

Empty Category Principle (ECP)

We have already defined 'government'. Now, let us define 'proper government':

α properly governs β iff:
α governs β and α is lexical (i.e. N, V, A, or P) or
α locally $\bar{A}$-binds β

The ECP says 'A trace must be properly governed'.

This principle justifies the creation of empty categories during NP-trace and wh-trace and also explains the subject/object asymmetries to some extent. As in the following sentences:

(a) What_i do you think that Mukesh ate e_i?
(b) What_i do you think Mukesh ate e_i?

Bounding and Control Theory

There are many other types of constraints on Move α. It is not possible to explain all of them here, for details, see Peter Sells (1985).

In English, the long distance movement for complement clause can be explained by bounding theory if NP and S are taken to be bounding nodes. The theory says that the application of Move α may not cross more than one bounding node. The theory of control involves syntax, semantics, and pragmatics. As stated previously, the empty category 'PRO' (+a, +p) behaves as an anaphor sometimes, when it is the subject of the clausal complement to verbs such as decide and try. However, it behaves as pronoun with some other verbs.

Case Theory and Case Filter

In GB, case theory deals with the distribution of NPs and mentions that each NP (with the possible exception of a few empty categories) must be assigned a case. In English, we have the nominative, objective, genitive, etc., cases, which are assigned to NPs at particular positions. Indian languages are rich in case-markers, which are carried even during movements.

Case Filter An NP is ungrammatical if it has phonetic content or if it is an argument (with the exception of big 'PRO') and is not case-marked. Phonetic content here, refers to some physical realization, as opposed to empty categories. Thus, case filters restrict the movement of NP at a position which has no case assignment. It works in a manner similar to that of the θ-criterion.

In short, GB presents a model of the language which has three levels of syntactic representation. It assumes phrase structures to be the maximal projection of some lexical head and in a similar fashion, explains the structure of a sentence or a clause. It assigns various types of roles to these structures and allows them a broad kind of movement called Move α. It then defines various types of constraints which restrict certain movements and justifies others. GB gives a new insight for the modelling of languages, although the Chomskian Minimalist Programme has superseded GB.

2.2.4 Lexical Functional Grammar (LFG) Model

This section presents those features of LFG that throw a light on language modelling. For the details of lexical functional grammar, readers are encouraged to seek Darlymple et al. (1995).

Unlike GB, LFG represents sentences at two syntactic levels—constituent structure (c-structure) and functional structure (f-structure). Based on Woods' *Augmented Transition Networks* 1970, which used phrase structure trees to represent the surface structure of sentences and the underlying predicate–argument structure, Kaplan (1975a, b) proposed a concrete form for the register names and values (used in ATN implementation), which became the functional structures in LFG. On the other hand, Bresnan (1976a, 1977) was more concerned with the problem of explaining some linguistic issues, such as active/passive and dative alternations, in transformational approach. She proposed that such issues can be dealt with by using lexical redundancy rules. The unification of these two diverse approaches (with a common concern) led to the development of the LFG theory, which was presented as *Lexical Functional Grammar: A Formal System for Grammatical Representation* in 1982.

The LFG is a formalism that is both computationally and linguistically motivated and provides precise algorithms for linguistic issues it can handle. The term 'lexical functional' is composed of two terms: the 'functional' part is derived from 'grammatical functions', such as subject and object, or roles played by various arguments in a sentence. The 'lexical' part is derived from the fact that the lexical rules can be formulated to help define the given structure of a sentence and some of the long distance dependencies, which is difficult in transformational grammars.

C-structure and f-structure in LFG

As LFG is aimed at providing exact computational algorithms, it provides well-defined objects called constituent structure (c-structure) and functional structure (f-structure). The c-structure is derived from the usual phrase and sentence structure syntax, as in CFG (discussed in Chapter 4). However,

as the grammatical-functional role cannot be derived directly from phrase and sentence structure, functional specifications are annotated on the nodes of c-structure, which when applied on sentences, results in f-structure. Hence, f-structure is the final product which encodes the information obtained from phrase and sentence structure rules and functional specifications.

Let us consider an example.

Example 2.5

She saw stars in the sky.

CFG rules to handle this sentence are:

S → NP VP
VP → V {NP} {NP} PP* {S'}
PP → P NP
NP → Det N {PP}
S' → Comp S

where

S: sentence		V:	verb
P: preposition		N:	noun
S': clause		Comp:	complement

{ } optional

*: Phrase can appear any number of times including blank.

When annotated with functional specifications, the rules become:

Rule 1: $S \rightarrow$ *NP* *VP*
↑ subj = ↓ ↑ = ↓

Rule 2: $VP \rightarrow V$ {*NP*} {*NP*} *PP** {*S'*}
↑ obj = ↓ ↑ obj 2 = ↓ ↑ (↓ case) = ↓ ↑ comp = ↓

Rule 3: $PP \rightarrow P$ *NP*
↑ obj = ↓

Rule 4: $NP \rightarrow$ {Det} *N* {*PP*}
↑ Adjunct = ↓

Rule 5: $S' \rightarrow$ Comp *S*
↑ = ↓

Here, ↑ (up arrow) refers to the f-structure of the mother node that is on the left hand side of the rule. The ↓ (down arrow) symbol refers to the f-structure of the node under which it is denoted.

Hence, in Rule 1, (↑ subj = ↓) indicates that the f-structure of the first NP goes to the f-structure of the subject of the sentence, while (↑ = ↓) indicates that the f-structure of the VP node goes directly to the f-structure

of the sentence VP. Similarly, in Rule 2, the f-structure of VP is defined by the lexical item V, the two optional NPs, any number of PPs, and the optional clause(S'). The f-structure of V can be obtained from the lexicon itself. All terminals in LFG can be thought of as annotated with ↑ = ↓. The NPs can function as object and object 2 of the sentence, and their f-structures are obained using f-sructure of Obj and Obj_2. '↑ (↓case) = ↓' in rule 2 indicaes that the f-sructure of the PP and the case of PP (some literature refers it as P case) determines the f-structure of VP. 'Comp' refers to the compliment in a sentence, e.g., 'He said *that* she is powerful'.

Let us see first the lexical entries of various words in the sentence. (2.4)

She saw stars.

She N (↑ Pred) = 'PRO'
(↑ Pers) = 3
(↑ Num) = SG
(↑ Gen) = FEM
(↑ Case) = NOM

Saw V ↑ Pred = 'see < (↑ Subj) (↑ Obj) >'
(↑ Tense = PAST)

Stars N ↑ Pred = 'Star'
↑ Pers = 3
↑ Num = PL

This will lead to the c-structure shown in Figure 2.8.

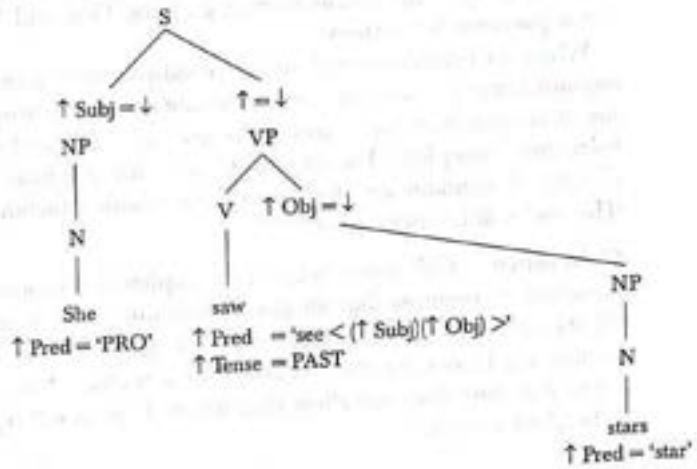


Figure 2.8 C-structure of sentence (2.4)

Finally, the f-structure is the set of attribute–value pairs, represented as

$$\begin{bmatrix} \text{subj} & \begin{bmatrix} \text{Pers} & 3 \\ \text{Num} & \text{SG} \\ \text{Gen} & \text{FEM} \\ \text{Case} & \text{NOM} \\ \text{Pred} & \text{'PRO'} \end{bmatrix} \\ \text{obj} & \begin{bmatrix} \text{Pers} & 3 \\ \text{Num} & \text{PL} \\ \text{Pred} & \text{'Star'} \end{bmatrix} \\ \text{Pred} & \text{'see'} <(\uparrow \text{subj})\ (\uparrow \text{obj})> \end{bmatrix}$$

It is interesting to note that the final f-structure is obtained through the unification of various f-structures for subject, object, verb, complement, etc. This unification is based on the functional specifications of the verb, which predicts the overall sentence structure.

LFG requires that all possible structures corresponding to passive constructs, dative constructs, etc., must be specified. If the given sentence does not match the specifications, it is said to be ill-formed. LFG imposes three conditions on f-structure (Sells 1985).

Consistency In a given f-structure, a particular attribute may have at the most one value. Hence, while unifying two f-structures, if the attribute Num has value SG in one and PL in the other, it will be rejected.

Completeness A function is called governable if it appears within the Pred value of some lexical form, e.g., Subj, Obj, and Obj 2. Adjunct is not a governable function.

When an f-structure and all its subsidiary f-structures (as the value of any attribute of f-structure can again contain other f-structures) contain all the functions that their predicates govern, then and only then is the f-structure complete. For example, since the predicate 'see < (↑ Subj) (↑ Obj) >' contains an object as its governable function, a sentence like 'He saw' will be incomplete.

Coherence Coherence maps the completeness property in the reverse direction. It requires that all governable functions of an f-structure, and all its subsidiary f-structures, must be governed by their respective predicates. Hence, in the f-structure of a sentence, an object cannot be taken if its verb does not allow that object. Thus, it will reject the sentence, 'I laughed a book.'

The completeness and coherence conditions are counterparts of θ-criterion in GB theory.

Lexical Rules in LFG

Different theories have different kinds of lexical rules and constraints for handling various sentence-constructs (active, passive, dative, causative, etc.). In GB, to express a sentence in its passive form, the verb is changed to its participial form and the ability of the verb to assign case and external (Agent) θ-role is taken away. In LFG, the verb is converted to the participial form, but the sub-categorization is changed directly. Consider the following example:

Active: Tara ate the food.
Passive: The food was eaten by Tara.
Active: ↑ Pred = 'eat < (↑ Subj) (↑ Obj) >'
Passive: ↑ Pred = 'eat < (↑ Obl_{ag}) (↑ Subj) >'

Here, Obl_{ag} represents oblique agent phrase. Similar rules can be applied in active and dative constructs for the verbs that accept two objects.

Active: Tara gave a pen to Monika.
Passive: Tara gave Monika a pen.
Active: ↑ Pred = 'give < (↑ Subj) (↑ Obj_2) (↑ Obj) >'
Passive: ↑ Pred = 'give < (↑ Subj) (↑ Obj) (↑ Obl_{go})>'

Here, Obl_{go} stands for oblique goal phrase. Similar rules are also applicable to the process of causativization. This can be seen in Hindi, where the verb form is changed as follows:

हँसना	$\xrightarrow{\text{causativization}}$	हँसाना
Laugh		Laugh-cause-past made to laugh

Example 2.6

Active: तारा हँसी
Taaraa hansii
Tara laughed

Causative: मोनिका ने तारा को हँसाया
Monika ne Tara ko hansaayaa
Monika Subj Tara Obj laugh-cause-past
Monika made Tara to laugh.

Active: ↑ Pred = 'Laugh <↑ Subj>'
Causative: ↑ Pred = 'cause <(↑ Subj) (↑ Obj) (Comp)>'

Finally, the f-structure is the set of attribute–value pairs, represented as

$$\begin{bmatrix} \text{subj} & \begin{bmatrix} \text{Pers} & 3 \\ \text{Num} & \text{SG} \\ \text{Gen} & \text{FEM} \\ \text{Case} & \text{NOM} \\ \text{Pred} & \text{'PRO'} \end{bmatrix} \\ \text{obj} & \begin{bmatrix} \text{Pers} & 3 \\ \text{Num} & \text{PL} \\ \text{Pred} & \text{'Star'} \end{bmatrix} \\ \text{Pred} & \text{'see'} <(\uparrow \text{subj})\ (\uparrow \text{obj})> \end{bmatrix}$$

It is interesting to note that the final f-structure is obtained through the unification of various f-structures for subject, object, verb, complement, etc. This unification is based on the functional specifications of the verb, which predicts the overall sentence structure.

LFG requires that all possible structures corresponding to passive constructs, dative constructs, etc., must be specified. If the given sentence does not match the specifications, it is said to be ill-formed. LFG imposes three conditions on f-structure (Sells 1985).

Consistency In a given f-structure, a particular attribute may have at the most one value. Hence, while unifying two f-structures, if the attribute Num has value SG in one and PL in the other, it will be rejected.

Completeness A function is called governable if it appears within the Pred value of some lexical form, e.g., Subj, Obj, and Obj 2. Adjunct is not a governable function.

When an f-structure and all its subsidiary f-structures (as the value of any attribute of f-structure can again contain other f-structures) contain all the functions that their predicates govern, then and only then is the f-structure complete. For example, since the predicate 'see < (↑ Subj) (↑ Obj) >' contains an object as its governable function, a sentence like 'He saw' will be incomplete.

Coherence Coherence maps the completeness property in the reverse direction. It requires that all governable functions of an f-structure, and all its subsidiary f-structures, must be governed by their respective predicates. Hence, in the f-structure of a sentence, an object cannot be taken if its verb does not allow that object. Thus, it will reject the sentence, 'I laughed a book.'

The completeness and coherence conditions are counterparts of θ-criterion in GB theory.

Lexical Rules in LFG

Different theories have different kinds of lexical rules and constraints for handling various sentence-constructs (active, passive, dative, causative, etc.). In GB, to express a sentence in its passive form, the verb is changed to its participial form and the ability of the verb to assign case and external (Agent) θ-role is taken away. In LFG, the verb is converted to the participial form, but the sub-categorization is changed directly. Consider the following example:

Active: Tara ate the food.
Passive: The food was eaten by Tara.
Active: ↑ Pred = 'eat < (↑ Subj) (↑ Obj) >'
Passive: ↑ Pred = 'eat < (↑ Obl_{ag}) (↑ Subj) >'

Here, Obl_{ag} represents oblique agent phrase. Similar rules can be applied in active and dative constructs for the verbs that accept two objects.

Active: Tara gave a pen to Monika.
Passive: Tara gave Monika a pen.
Active: ↑ Pred = 'give < (↑ Subj) (↑ Obj_2) (↑ Obj) >'
Passive: ↑ Pred = 'give < (↑ Subj) (↑ Obj) (↑ Obl_{go})>'

Here, Obl_{go} stands for oblique goal phrase. Similar rules are also applicable to the process of causativization. This can be seen in Hindi, where the verb form is changed as follows:

हँसना —causativization→ हँसाना
Laugh — Laugh-cause-past
made to laugh

Example 2.6

Active: तारा हँसी
Taaraa hansii
Tara laughed

Causative: मोनिका ने तारा को हँसाया
Monika ne Tara ko hansaayaa
Monika Subj Tara Obj laugh-cause-past
Monika made Tara to laugh.

Active: ↑ Pred = 'Laugh <↑ Subj>'
Causative: ↑ Pred = 'cause <(↑ Subj) (↑ Obj) (Comp)>'

Here, a new predicate is formed which causes the action and requires a new subject, while the old subject becomes the object of the new predicate and the old verb becomes the X-complement (complement to infinitival VPs).

Long Distance Dependencies and Coordination

In GB, when a category moved, it creates an empty category. In LFG, unbounded movement and coordination is handled by the functional identity and by correlation with the corresponding f-structure. An example will better explain these ideas.

Example 2.7 Consider the wh-movement in the following sentence.

Which picture does Tara like—most?

The f-structure can be represented as follows:

$$\begin{bmatrix} \text{Focus} & \begin{bmatrix} \text{Obl}_{\perp} & \begin{bmatrix} \text{pred} & \text{'PRO'} \\ \text{Refl} & + \end{bmatrix} \\ \text{Pred} & \text{'picture}\ (\langle \text{Obl}_{\perp} \rangle)\text{'} \end{bmatrix} \\ \text{Subj} & [\text{pred}\ \text{'Tara'}] \\ \text{Obj} & [\quad\quad] \\ \text{Pred} & \text{'like}\ (\langle \uparrow\text{Subj} \rangle\ \langle \uparrow\text{Obj} \rangle)\text{'} \end{bmatrix}$$

The mechanism of handling these movements and coordination are not detailed here. The only aim is to highlight efforts and issues involved in modelling language.

2.2.5 Paninian Framework

Another very important model which has drawn much attention is the Paninian Grammar-based model (Kiparsky 1982, Bharti et al. 1995). Although *Paninian grammar* (PG) was written by Panini in 500 BC in Sanskrit (the original text being titled *Asthadhyayi*), the framework can be used for other Indian languages and possibly some Asian languages as well.

Unlike English, Asian languages are SOV (Subject-Object-Verb) ordered and inflectionally rich. The inflections provide important syntactic and semantic cues for language analysis and understanding. The Paninian framework takes advantage of these features. However, it should be noted that the research on this framework is still in progress and there are many complexities of Indian languages which are yet to be explained through this or other models. In this section, we briefly discuss some unique features of PG, to provide a glimpse of another potential model.

Some Important Features of Indian Languages

Indian languages have traditionally used oral communication for knowledge propagation. The purpose of these languages is to communicate ideas from the speaker's mind to the listener's mind. Such oral traditions have given rise to a morphologically rich language. Also, they are relatively word-order free. Some languages, like Sanskrit, have the flexibility to allow word groups representing subject, object, and verb to occur in any order. In others, like Hindi, we can change the position of subject and object. For example:

(a) माँ बच्चे को खाना देती है।
Maan Bachche ko khanaa detii hai
Mother child to food give–(s)
Mother gives food to the child.

(b) बच्चे को माँ खाना देती है।
Bachche ko Maan khanaa detii hai
Child to mother food give–(s)
Mother gives food to the child.

The auxilary verbs follow the main verb. In Hindi, they remain as separate words, whereas in south Indian (Dravidian) languages, they combine with the main verb. For example:

खा रहा है	करता रहा है
khaa raha hai	*kartaa rahaa hai*
eat-ing	doing been has
eating	has been doing

In Hindi, some verbs (main), e.g., give (देना), take (लेना), also combine with other verbs (main) to change the aspect and modality of the verbs.

Example 2.8

उसने खाना खाया।	उसने खाना खा लिया।
Usne khanaa khaayaa	*Usne khaanaa kha liyaa*
He (Subj) food ate	He (Subj) food eat taken
He ate food	He ate food (completed the action)
वह चला	वह चल दिया
	He move given
He moved	He moved (started the action)

In Indian languages, the nouns are followed by post-positions instead of prepositions. They generally remain as separate words in Hindi, except in the case of pronouns, for example

रेखा के पिता	उसके पिता
Rekha ke pita	*Uske pita*
Rekha of father	
Father of Rekha	Her (His) father

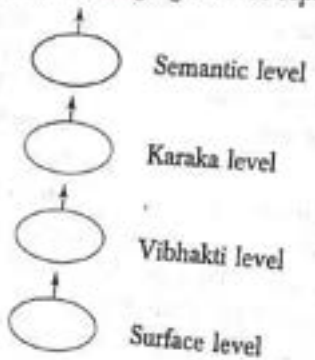
In view of such differences between English (and English-like languages) and Indian languages, it is imperative that we find a new framework for handling Indian languages. Even among Indian languages, all features are not the same. As noted earlier, verb groups are formed differently in Indo-Aryan and Dravidian languages. Sanskrit is very different from the other Indian languages as it has five tenses and three numbers, and only one time aspect in each tense. Hence, the translation of 'He goes' and 'He is going' is the same in Sanskrit. Hindi is unique in the sense that it has no neuter gender. All nouns are categorized as feminine or masculine, and the verb form must have a gender agreement with the subject (sometimes with the object).

Thus, we have

ताला खो गया	चाभी खो गयी
Taalaa kho gayaa	*Chaabhii kho gayee*
Lock lose (past)	key lose (past)
The lock was lost.	The key was lost.

Layered Representation in PG

The GB theory represents three syntactic levels: deep structure, surface structure, and logical form (LF), where the LF is nearer to semantics. This theory tries to resolve all language issues at syntactic levels only. Unlike GB, Paninian grammar framework is said to be syntactico-semantic, that is, one can go from surface layer to deep semantics by passing through intermediate layers. Although all these layers are not named, as per Bharti, et al. (1995), the language can be represented as follows:



The surface and the semantic levels are obvious. The other two levels should not be confused with the levels of GB. *Vibhakti* literally means inflection, but here, it refers to word (noun, verb, or other) groups based either on case endings, or post-positions, or compound verbs, or main and auxiliary verbs, etc. Instead of talking about NP, VP, AP, PP, etc., word groups are formed based on various kinds of markers (including the absence of it or θ). These markers are language-specific, but all Indian languages (and possibly Asian languages as well) can be represented at the Vibhakti level.

Karaka (pronounced *Kaaraka*) literally means Case, and in GB, we have already discussed case theory, θ-theory, and sub-categorization, etc. Paninian Grammar has its own way of defining Karaka relations, which we discuss in the next section. These relations are based on the way the word groups participate in the activity denoted by the verb group. In this sense, it is semantic as well as syntactic. However, things are not so straightforward. Complexities arise because of the absence of inflections, multiple categories of the words, multiple meanings, and above all, the presence of a large number of exceptions. These exceptions are not only applicable on stated rules but also on future rules. Such forward and backward chaining makes actual implementation difficult.

As the purpose of these languages is to communicate, generally between one human and another, the resolution of ambiguities is a contentious issue, often left to the listener. Hence, there may not be any particular number of semantic levels. Multiple-meaning texts are abundant in Indian literature as seen in the hundreds of interpretations of the epics.

Karaka Theory

Karaka theory is the central theme of PG framework. Karaka relations are assigned based on the roles played by various participants in the main activity. These roles are reflected in the case markers and post-position markers (parsargs). These relations are similar to case relations in English, but the types of relations are defined in a different manner and the richness of the case endings found in Indian languages has been used to its advantage.

We will discuss the various Karakas, such as Karta (subject), Karma (object), Karana (instrument), Sampradana (beneficiary), Apadan (separation), and Adhikaran (locus). These descriptions are just examples and not a complete discussion of PG or Karaka theory. For details of Karaka theory, see Shastri (1973) and Vasu (1977).

To explain various Karaka relations, let us consider an example. Consider the following Hindi sentence:

माँ बच्ची को आँगन में हाथ से रोटी खिलाती है। (2.5)

Maan bachchi ko aangan mein haath se rotii khilaatii hei

Mother child-to courtyard-in hand-by bread feed (s).

The mother feeds bread to the child by hand in the courtyard.

The first important Karak is subject, called 'Karta' in PG. Karta is defined as the noun group which is most independent (*swatantra* in Hindi). Karta has generally 'ne' or 'ϕ' case marker. It is an independent entity in the activity denoted by the main verb. As seen in GB also, verbs do not sub-categorize for subjects, although they assign a θ role to it. In sentence 2.5, 'maan' (mother) is the Karta. The concept of Karta is different from the 'agent' concept in the sense that Karta can also take up the role of experiencer, e.g.,

मुझसे रहा न गया।

Mujhse rahaa na gayaa

Me hold -not -passive

I could not hold myself.

'Karma' is similar to object and is the locus of the result of the activity. In sentence (2.5), *rotii* (bread) is the Karma. As explained earlier, when the Karta is the experiencer, it (she) is also the locus of the result. Thus, the locus of the result is only when it is different from Karta termed Karma. Karma generally has 'ϕ' or 'KO' case marker. Another Karaka relation is 'Karan' (instrument), which is a noun group through which the goal is achieved. In sentence (2.5), *haath* (hand) is the Karan. It has the marker *dwara* (by) or *se*. 'Sampradan' is the beneficiary of the activity, e.g., *bachchi* (child) in sentence (2.5). It takes the marker *ko* (to) or *ke liye* (for). 'Apaadaan' denotes separation and the marker is attached to the part that serves as a reference point (being stationary), for example

माँ ने थाली से खाना उठाकर बच्चे को दिया।

Maan ne thaali se khana uthakar bachche ko diyaa

Mother-Karta plate from Apaadan food taking-up child-to gave.

The mother gave food to the child taking it up from the plate.

Here *thaali* is the Apaadan. 'Adhikaran' is the locus (support in space or time) of Karta or Karma. In sentence (2.5), *aangan* (courtyard) is the Adhikaran. As these six relations are not sufficient to capture all possible relations, various others such as 'Sambandh' (relation) and 'Tadarthya' (purpose) have also been tried.

Issues in Paninian Grammar

The two problems challenging linguists are:

(i) Computational implementation of PG, and

(ii) Adaptation of PG to Indian, and other similar languages.

An approach to implementing PG has been discussed in Bharati, et al. (1995). This is a multilayered implementation. The approach is named 'Utsarga-Apvada' (default-exception), where rules are arranged in multiple layers in such a way that each layer consists of rules which are in exception to rules in the higher layer. Thus, as we go down the layer, more particular information is derived. Rules may be represented in the form of charts (such as Karaka chart and Lakshan chart).

However, many issues remain unresolved, specially in cases of shared Karak relations. Another difficulty arises when mapping between the Vibhakti (case markers and post-positions) and the semantic relation (with respect to verb) is not one to one. Two different Vibhakti can represent the same relation, or the same Vibhakti can represent different relations in different contexts. The strategy to disambiguate the various senses of words, or word groupings, are still the challenging issues.

As the system of rules is different in different languages, the framework requires adaptations to tackle various applications in various languages. Only some general features of PG framework has been described here.

2.3 STATISTICAL LANGUAGE MODEL

A statistical language model is a probability distribution $P(s)$ over all possible word sequences (or any other linguistic unit like words, sentences, paragraphs, documents, or spoken utterances). A number of statistical language models have been proposed in literature. The dominant approach in statistical language modelling is the *n*-gram model.

2.3.1 *n*-gram Model

As discussed earlier, the goal of a statistical language model is to estimate the probability (likelihood) of a sentence. This is achieved by decomposing sentence probability into a product of conditional probabilities using the chain rule as follows:

$$P(s) = P(w_1, w, w_3, \ldots, w_n)$$
$$= P(w_1)\, P(w_2/w_1)\, P(w_3/w_1\, w_2)\, P(w_4/w_1\, w_2\, w_3) \ldots P(w_n/w_1\, w_2 \ldots w_{n-1}))$$

$$= \prod_{i=1}^{n} P(w_i/h_i)$$

where h_i is history of word w_i defined as

$$w_1 w_2 \ldots w_{i-1}$$

So, in order to calculate sentence probability, we need to calculate the probability of a word, given the sequence of words preceding it. This is not a simple task. An n-gram model simplifies the task by approximating the probability of a word given all the previous words by the conditional probability given previous n-1 words only.

$$P(w_i/h_i) = P(w_i/w_{i-n+1} \ldots w_{i-1})$$

Thus, an n-gram model calculates $P(w_i/h_i)$ by modelling language as Markov model of order $n-1$, i.e., by looking at previous n-1 words only. A model that limits the history to the previous one word only, is termed a bi-gram ($n = 1$) model. Likewise, a model that conditions the probability of a word to the previous two words, is called a tri-gram ($n = 2$) model. Using bi-gram and tri-gram estimate, the probability of a sentence can be calculated as:

$$P(s) = \prod_{i=1}^{n} P(w_i/w_{i-1})$$

and $$P(s) = \prod_{i=1}^{n} P(w_i/w_{i-2} \cdot w_{i-1})$$

As an example, the bi-gram approximation of P(east/The Arabian knights are fairy tales of the) is

P(east/the),

whereas a tri-gram approximation is

P(east/of the).

A special word (pseudo word) <s> is introduced to mark the beginning of the sentence in bi-gram estimation. The probability of the first word in a sentence is conditioned on <s>. Similarly, in tri-gram estimation, we introduce two pseudo-words <s1> and <s2>.

Now, we discuss how to estimate these probabilities. This is done by training the n-gram model on the training corpus. We estimate n-gram parameters using the maximum likelihood estimation (MLE) technique, i.e., using relative frequencies. We count a particular n-gram in the training corpus and divide it by the sum of all n-grams that share the same prefix.

$$P(w_i/w_{i-n+1}, \ldots, w_{i-1}) = \frac{C(w_{i-n+1}, \ldots, w_{i-1}, w_i)}{\sum_{w} C(w_{i-n+1}, \ldots, w_{i-1}, w)}$$

The sum of all n-grams that share first n-1 words is equal to the count of the common prefix $w_{i-n+1}, \ldots, w_{i-1}$. So, we rewrite the previous expression as follows:

$$P(w_i/w_{i-n+1}, \ldots, w_{i-1}) = \frac{C(w_{i-n+1}, \ldots, w_{i-1}, w_i)}{C(w_{i-n+1}, \ldots, w_{i-1})}$$

The model parameter we get using these estimates, maximizes the probability of the training set T given the model M, i.e., $P(T/M)$. The frequency with which a word occurs in a text may not be the same as in the training set; this model only provides the most likely solution.

A number of improvements have been suggested for the standard n-gram model. Before we discuss them, let us illustrate these ideas with the help of an example.

Example 2.9

Training set:

The Arabian Knights
These are the fairy tales of the east
The stories of the Arabian knights are translated in many languages

Bi-gram model:

P(the/<s>) = 0.67	P(Arabian/the) = 0.4	P(knights /Arabian) = 1.0
P(are/these) = 1.0	P(the/are) = 0.5	P(fairy/the) = 0.2
P(tales/fairy) = 1.0	P(of/tales) = 1.0	P(the/of) = 1.0
P(east/the) = 0.2	P(stories/the) = 0.2	P(of/stories) = 1.0
P(are/knights) = 1.0	P(translated/are) = 0.5	P(in /translated) = 1.0
P(many/in) = 1.0		
P(languages/many) = 1.0		

Test sentence(s): The Arabian knights are the fairy tales of the east.

P(The/<s>) × P(Arabian/the) × P(Knights/Arabian) × P(are/knights) × P(the/are) × P(fairy/the) × P(tales/fairy) × P(of/tales) × P(the/of) × P(east/the)

$= 0.67 \times 0.5 \times 1.0 \times 1.0 \times 0.5 \times 0.2 \times 1.0 \times 1.0 \times 1.0 \times 0.2$

$= 0.0067$

As each probability is necessarily less than 1, multiplying the probabilities might cause a numerical underflow, particularly in long sentences. To avoid this, calculations are made in log space, where a calculation corresponds to adding log of individual probabilities and taking antilog of the sum.

The n-gram model suffers from data sparseness problem. An n-gram that does not occur in the training data is assigned zero probability, so that even a large corpus has several zero entries in its bi-gram matrix. This is because of the assumption that the probability of occurrence of a word depends only on the preceding word (or preceding $n-1$ words), which is not true in general. There are several long distance dependencies in natural language sentences, which this model fails to capture. Goodman (2003) pointed out that 'there is rarely enough data to accurately estimate the parameters of a language model.'

A number of smoothing techniques have been developed to handle the data sparseness problem, the simplest of these being add-one smoothing. In the words of Jurafsky and Martin (2000):

> *Smoothing in general refers to the task of re-evaluating zero-probability or low-probability n-grams and assigning them non-zero values.*

The word 'smoothing' is used to denote these techniques because they tend to make distributions more uniform by moving the extreme probabilities towards the average.

2.3.2 Add-one Smoothing

This is the simplest smoothing technique. It adds a value of one to each n-gram frequency before normalizing them into probabilities. Thus, the conditional probability becomes:

$$P(w_i/w_{i-n+1}, \ldots, w_{i-1}) = \frac{C(w_{i-n+1}, \ldots, w_{i-1}, w_i)}{C(w_{i-n+1}, \ldots, w_{i-1}) + V}$$

where V is the vocabulary size, i.e., size of the set of all the words being considered.

In general, add-one smoothing is not considered a good smoothing technique. It assigns the same probability to all missing n-grams, even though some of them could be more intuitively appealing than others. Gale and Church (1994) reported that variance of the counts produced by the add-one smoothing is worse than the unsmoothed MLE method. Another problem with this technique is that it shifts too much of the probability mass towards the unseen n-grams (n-grams with 0 probabilities) as there number is usually quite large. Good–Turing smoothing (Good 1953) attempts to improve the situation by looking at the number of n-grams with a high frequency in order to estimate the probability mass that needs to be assigned to missing or low-frequency n-grams.

2.3.3 Good–Turing Smoothing

Good–Turing smoothing (Good 1953) adjusts the frequency f of an n-gram using the count of n-grams having a frequency of occurrence $f+1$. It converts the frequency of an n-gram from f to f^* using the following expression:

$$f^* = (f+1)\frac{n_{f+1}}{n_f}$$

where n_f is the number of n-grams that occur exactly f times in the training corpus. As an example, consider that the number of n-grams that occur 4 times is 25,108 and the number of n-grams that occur 5 times is 20,542. Then, the smoothed count for 5 will be

$$\frac{20542}{25108} \times 5 = 4.09$$

2.3.4 Caching Technique

Another improvement over basic n-gram model is caching. The frequency of n-gram is not uniform across the text segments or corpus. Certain words occur more frequently in certain segments (or documents) and rarely in others. For example, in this section, the frequency of the word 'n-gram' is high, whereas it occurs rarely in earlier sections. The basic n-gram model ignores this sort of variation of n-gram frequency. The cache model combines the most recent n-gram frequency with the standard n-gram model to improve its performance locally. The underlying assumption here is that the recently discovered words are more likely to be repeated.

A number of other smoothing techniques appear in literature. For these, readers are referred to Chen and Goodman (1998).

SUMMARY

The main topics covered in this chapter are as follows.

- Language modelling deals with providing a description of natural languages amenable to processing.
- There are two main approaches to language modelling: grammar-based language model and statistical language model.
- A grammar-based language model uses grammar to model language. A number of computational grammars have been proposed in literature. This chapter provides a discussion of models based on

government and binding, lexical functional grammar, and Paninian grammar.

- GB theory talks about representations at four different levels: s-structure, d-structure, logical form, and phonetic form.
- An important concept of GB is a general transformational rule called Move α which moves constituents freely, subject to certain constraints (defined by several theories and principles).
- LFG represents sentences at two different levels—constituent structure (c-structure) and functional structure (f-structure).
- Paninian grammar provides a framework for modelling Indian languages.
- The Paninian framework is syntactico-semantic. The central theme of this framework is Karaka relations.
- A statistical language model estimates the probability (likelihood) of a sentence. The most widely used statistical model is n-gram model.
- The *n*-gram model suffers from sparseness of data. Smoothing techniques such as add-one and Good–Turing can be used to handle this problem.
- Caching is another improvement over the standard *n*-gram model.

REFERENCES

Bharati, A., V. Chaitanya, and R. Sangal, 1995, 'Natural Language Processing: A Paninian Perspective,' Prentice-Hall of India, New Delhi.

Bresnan, Joan, 1976, 'Evidence for a theory of unbounded transformation,' *Linguistic Analysis*.

———1977, 'Variables in theory of transformations,' *Formal Syntax*, Peter W. Culicover, Thoman Wasow, and Adrian Akmajian (Eds.), Academic Press, New York, pp. 157–96.

Chen, Stanley F. and Joshua T. Goodman, 1998, 'An Empirical Study of Smoothing Techniques for Language Modelling,' *Technical Report TR-10-98*, Computer Science Group, Harvard University.

Chomsky, Noam, 1957, *Syntactic Structures*, Mouton, The Hague.

———1986, *Some Concepts and Consequences of the Theory of Government and Binding*, MIT Press, Cambridge, MA.

Gale, William A. and Kenneth W. Church, 1994, 'What's wrong with adding one?,' Corpus-based Research into Language, Oostdijk and P. de Haan (Eds.), Rodolpi, Amsterdam.

Gazdar, G., E. Klein, G. Pullum, and I. Sag, 1985, *Generalized Phrase Structure Grammar*, Blackwell.

Good, I.J., 1953, 'The population frequencies of species and the estimation of population parameters,' *Biometrika*, 40(3 and 4), pp. 237–64.

Goodman, Joshua, 2003, 'The state of the art in language modelling,' *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology: Tutorials*, 5, Edmonton, Canada.

Jurafsky, Daniel and James H. Martin, 2000, 'Speech and Language Processing: An Introduction to Natural Language Processing,' *Computational Linguistics and Speech Recognition*, Prentice Hall, New Jersey.

Kaplan, Ronald M., 1975a, 'On Process Models for Sentence Comprehension,' *Explorations in Cognition*, Donald A. Norman and David E. Rumelhart (Eds.), W.H. Freeman, San Francisco.

———1975b, 'Transient Processing Load in Relative Clauses,' *Doctoral Dissertation*, Harvard University.

———1995, 'The Formal Architecture of Lexical-Functional Grammar,' *Formal Issues in Lexical-Functional Grammar*, M. Dalrymple, R.M. Kaplan, J.T. Maxwell III, and A. Zaenen (Eds.), CSLI Publications, Stanford.

Kaplan, Ronald M. and Joan Bresnan, 1982, *The Mental Representation of Grammatical Relations*, Joan Bresnan (Ed.), The MIT Press, Cambridge, MA.

Kiprasky, P., 1982, 'Some theoretical problem in Panini's grammar,' Bhandarkar Oriental Research Institute, Poona.

Rosenfeld, Ronald, 1994, 'Adaptive Statistical Language Modelling: A Maximum Entropy Approach,' *D.Phil. Thesis*, Carnegie Mellon University, Pittsburgh.

Sells, Peter, 1985, 'Lectures on Contemporary Syntactic Theories,' Center for the Study of Language and Information (CSLI), *Lecture Notes*, Number 3, Stanford.

Shastri, Charudev, 1973, *Vyakarana Chandrodaya*, (Vol. I–V), Motilal Banarsidass, in Hindi, New Delhi

Vasu, S.C. (Tr.), 1977, *The Ashtadhyayi of Panini* (2 volumes), Motilal Banarsidass, New Delhi.

Woods, William A., 1970, 'Transition Network Grammars for Natural Language Analysis,' *Communications of the ACM*, 13(10), pp. 591–606.

EXERCISES

1. Give the representation of a sentence in d-structure and s-structure in GB.
2. How is the structure of a sentence different from the structure of a phrase?

3. Discuss empty-category principle and give two examples of the creation of empty categories.
4. What is f-structure in LFG? What inputs to an algorithm create an f-structure?
5. Using the annotated rules in Example 2.5, find f-structures for the various phrases and the complete sentence, 'I saw Aparna in the market at night'.
6. What are Karaka relations? Explain the difference between Karta and Agent.
7. What are lexical rules? Give the complete entry for a verb in the lexicon, to be used in LFG.
8. Compare GB and PG. Why is PG called syntactico-semantic theory?
9. What are the problems associated with n-gram model? How are these problems handled?
10. How does caching improve the *n*-gram model?

LAB EXERCISES

1. Create a collection of 10 documents. Write a program to find the frequency of bi-grams in this collection.
2. Find the number of bi-grams that occur more than three times in this collection.

CHAPTER 3

WORD LEVEL ANALYSIS

CHAPTER OVERVIEW

This chapter focuses on processing carried out at word level, including methods for characterizing word sequences, identifying morphological variants, detecting and correcting misspelled words, and identifying correct part-of-speech of a word. The part-of-speech tagging methods covered in this chapter are: rule-based (linguistic), Stochastic (data-driven), and hybrid.

3.1 INTRODUCTION

As discussed in Chapter 1, natural language processing (NLP) involves different levels and complexities of processing. One way to analyse natural language text is by breaking it down into constituent units (words, phrases, sentences, and paragraphs) and then analyse these units. In Chapter 2, we discussed various language models that are used for analysing the syntax of natural language sentences. Before analysing syntax, we need to understand words, as words are the fundamental unit (syntactic as well as semantic) of any natural language text. This chapter focuses on NLP carried out at word level, including characterizing word sequences, identifying morphological variants, detecting and correcting misspelled words, and identifying correct part-of-speech of a word.

Regular expressions are a beautiful means for describing words. In many text applications, we wish to work with string patterns. Suppose you have just come across the word 'supernova'. It catches your interest and you jump to a search engine to find out more on 'supernovas'. But you do not know whether to type in 'supernova', 'Supernova', or 'supernovas'. Obviously, you need a system, which will retrieve relevant articles using any one of these word forms. Regular expressions are used for describing text strings in situations like this and in other information

retrieval applications. In this chapter, we introduce regular expressions and discuss standard notations for describing text patterns.

After defining regular expressions, we discuss their implementation using finite-state automaton (FSA). Readers who have gone through a course in formal language theory will be familiar with FSA. The FSAs and their variants, such as finite state transducers, have found useful applications in speech recognition and synthesis, spell checking, and information extraction. As we will be using FSA throughout this book, we formally define FSAs in this chapter.

Errors in typing and spelling are quite common in text processing applications. Detecting and correcting these errors are the next topics of discussion in this chapter. Numerous web pages also contain misspelled words and often, query terms entered into the search engines are misspelled. An interactive spelling facility that informs users of such errors and presents appropriate corrections to them, is useful in these applications.

There are different classes of words. A word has many forms and the same word may have many different meanings depending on the context. Identifying the class to which a word belongs, its basic form, and its meaning are crucial to analysing text. This is the last topic covered in this chapter.

3.2 REGULAR EXPRESSIONS

Regular expressions, or regexes for short, are a pattern-matching standard for string parsing and replacement. They are a powerful way to find and replace strings that take a defined format. For example, regular expressions can be used to parse dates, urls and email addresses, log files, configuration files, command line switches, or programming scripts. They are useful tools for the design of language compilers and have been used in NLP for tokenization, describing lexicons, morphological analysis, etc. We have all used simplified forms of regular expressions, such as the file search patterns used by MS DOS, e.g., dir*.txt.

The use of regular expressions in computer science was made popular by a Unix-based editor, 'ed'. Perl was the first language that provided integrated support for regular expressions. It used a slash around each regular expression; we will follow the same notation in this book. However, slashes are not a part of regular expressions.

Regular expressions were originally studied as a part of theory of computation. They were first introduced by Kleene (1956). A regular expression is an algebraic formula whose value is a pattern consisting of a

set of strings, called the language of the expression. The simplest kind of regular expression contains a single symbol. For example, the expression

/a/

denotes the set containing the string 'a'. A regular expression may specify a sequence of characters also. For example, the expression

/supernova/

denotes the set that contains the string 'supernova' and nothing else. In a search application, the first instance of each match to regular expression is underlined in Table 3.1.

Table 3.1 Some simple regular expressions

Regular expression	Example patterns
/book/	The world is a book, and those who do not travel read only one page.
/book/	Reporters, who do not read the stylebook, should not criticize their editors.
/face/	Not everything that is faced can be changed. But nothing can be changed until it is faced.
/a/	Reason, Observation, and Experience—the Holy Trinity of Science.

3.2.1 Character Classes

Characters are grouped by putting them between square brackets. This way, any character in the class will match one character in the input. For example, the pattern /[abcd]/ will match (any of) a, b, c, and d. The use of brackets specifies a disjunction of characters. The regular expression /[0123456789]/ specifies any single digit. The character classes are important building blocks in expressions. They sometimes lead to cumbersome notation. For example, it is inconvenient to write the regular expression

/[abcdefghijklmnopqrstuvwxyz]/

to specify 'any lowercase letter'. In these cases, a dash is used to specify a range. The regular expression /[5–9]/ specifies any one of the characters 5, 6, 7, 8, or 9. The pattern /[m–p]/ specifies any one of the letter m, n, o, or p.

Regular expressions can also specify what a single character cannot be, by the use of a caret at the beginning. For example, the pattern /[^x]/ matches any single character except x. This interpretation is true only when a caret appears as the first symbol. If it occurs at any other place, it refers to the caret symbol itself. Table 3.2 shows a few examples explaining these concepts.

Table 3.2 Use of square brackets

RE	Match	Example patterns matched
[abc]	Match any of a, b, and c	'Refresher course will start tomorrow'
[A–Z]	Match any character between A and Z (ASCII order)	'the course will end on Jan. 10, 2006'
[^A–Z]	Match any character other than an uppercase letter	'TREC Conference'
[^abc]	Match anything other than a, b, and c	'TREC Conference'
[+*?.]	Match any of +, *, ?, or the dot.	'3 + 2 = 5'
[a^]	Match a or ^	'^ has three different uses.'

Regular expressions are *case sensitive*. The pattern /s/ matches a lower case 's' but not an uppercase 'S'. This means that the pattern /sana/ will not match the string /Sana/. This problem can be solved by using the disjunction of character s and S. The pattern /[sS]/ will match the strings containing either 's' or 'S'. The pattern /[sS]ana/ matches with the string 'sana' or 'Sana'.

While the use of square brackets solves the capitalization problem, we still need a solution for how to specify both 'supernova' and 'supernovas'. The pattern /[sS]upernova[sS]/ matches with any of the strings 'supernovas', 'supernovas', 'Supernovas', and 'SupernovaS', but not with the string 'supernova'. This is achieved with the use of a question mark /?/. A question mark makes the preceding character optional, i.e., zero or one occurrence of the previous character. The regular expression

/supernovas?/

specifies both 'supernova' and 'supernovas'. Often, we need to specify repeated occurrences of a character. The * operator, called the *Kleene* * (pronounced 'cleany star'), allow us to do this. The * operator specifies zero or more occurrences of a preceding character or regular expression. The regular expression /b*/ will match any string containing zero or more occurrences of 'b', i.e., it will match 'b', 'bb', or 'bbbb'. It will also match 'aaa', since that string contains zero occurrences of 'b'. To match a string containing one or more 'b's, the regular expression is /bb*/. The regular expression /bb*/ means a 'b' followed by zero or more 'b's. This will match with any of the strings 'b', 'bb', 'bbb', 'bbbb'. Similarly, the regular expression /[ab]*/ specifies 'zero or more "a"s or "b"s. This will match strings like 'aa', 'bb', or 'abab'. The kleene+ provides a shorter notation to specify one or more occurrences of a character. The regular

expression /a+/ means one or more occurrences of 'a'. Using *Kleene+*, we can specify a sequence of digits by the regular expression /[0–9]+/. Complex regular expressions can be built up from simpler ones by means of regular expression operators.

The caret (^) is also used as an anchor to specify a match at the beginning of a line. The dollar sign, **$**, is an anchor that is used to specify a match at the end of the line. ^ and $ are important to regexes. If you wish to search for a line containing only the phrase 'The nature.' and nothing else, you need to specify a regular expression for this search. The anchors ^ and $ are of great help in this case. The regular expression /^The nature\.$/ will search exactly for this line.

A number of special characters are also used to build regular expressions. One such character is the dot (.), called wildcard character, which matches any single character. The wildcard expression /./ matches any single character. The regular expression /.at/ matches with any of the string cat, bat, rat, gat, kat, mat, etc. It will also match the meaningless words such as nat, 4at, etc. Table 3.3 shows some of the special characters and their likely use.

Table 3.3 Some special characters

RE	Description
.	The dot matches any single character.
\n	Matches a new line character (or CR+LF combination).
\t	Matches a tab (ASCII 9).
\d	Matches a digit [0–9].
\D	Matches a non-digit.
\w	Matches an alphanumeric character.
\W	Matches a non-alphanumberic character.
\s	Matches a whitespace character.
\S	Matches a non-whitespace character.
\	Use \ to escape special characters. For example, \. matches a dot, * matches a * and \\ matches a backslash.

We can also use the wildcard symbol for counting characters. For instance /.....berry/ matches ten-letter strings that end in berry. This finds patterns like strawberry, sugarberry, and blackberry but fails to match with blueberry or hackberry.

Suppose you are searching a text for the presence of 'Tanveer' or 'Siddiqui'. You cannot use square brackets for this. You need a disjunction operator, shown by the pipe symbol(|). The pattern blackberry|blackberries matches either 'blackberry' or 'blackberries'. You might prefer to do this

matching by merely writing blackberry|ies. Unfortunately, this does not work. The pattern blackberry|ies matches with either 'blackberry' or 'ies'. This is because sequences take precedence over disjunction. In order to apply the disjunction operator to a specific pattern, we need to enclose it within parentheses. The parenthesis operator makes it possible to treat the enclosed pattern as a single character for the purposes of neighboring operators. Now, we will consider an example from real application.

Example 3.1 Suppose we need to check if a string is an email address or not. An email address consist of a non-empty sequence of characters followed by the 'at' symbol, @, followed by another non-empty sequence of characters ending with pattern like .xx, .xxx, .xxxx, etc. The regular expression for an email address is

^[A-Za-z0-9_\.-]+@[A-Za-z0-9_\.-]+[A-Za-z0-9_][A-Za-z0-9_]$

Table 3.4 shows the various parts of this 'rgex'.

Table 3.4 Parts of regular expression of Example 3.1

Pattern	Description
^[A-Za-z0-9_\.-]+	Match a positive number of acceptable characters at the start of the string.
@	Match the @ sign.
[A-Za-z0-9_\.-]+	Match any domain name, including a dot.
[A-Za-z0-9_][A-Za-z0-9_]$	Match two acceptable characters but not a dot. This ensures that the email address ends with .xx, .xxx, .xxxx, etc.

This example works for most cases. However, the specification is not based on any standard and may not be accurate enough to match all correct email addresses. It may accept non-working email addresses and reject working ones. Fine-tuning is required for accurate characterization.

A regular expression characterizes a particular kind of formal language, called a regular language. The language of regular expressions is similar to formulas of Boolean logic. Like logic formulas, regular expressions represent sets. Regular language is set of strings described by the regular expression. Regular languages may be encoded as finite state networks.

A regular expression may contain symbol pairs. For example, the regular expression /a:b/ represents a pair of string. The regular expression /a:b/ actually denotes a regular relation. A regular relation may be viewed as a mapping between two regular languages. The a:b relation is simply the cross product of the languages denoted by the expressions /a/ and /b/. To differentiate the two languages that are involved in a regular relation, we call the first one, the upper, and the second one, the lower

language, of the relation. Similarly, in the pair /a:b/, the first symbol, a, can be called the upper symbol and the second symbol, b, the lower symbol. The two components of a symbol pair are separated in our notation by a colon (:) without any whitespace before or after. To make the notation less cumbersome, we ignore the distinction between the language A and the identity relation that maps every string of A to itself. Therefore, we also write /a:a/ simply as /a/.

Regular expressions have clean, declarative semantics (Voutilainen 1996). Mathematically, they are equivalent to finite automata, both having the same expressive power. This makes it possible to encode regular languages using finite-state automata, leading to easier manipulation of context free and other complex languages. Similarly, regular relations can be represented using finite-state transducers. With this representation, it is possible to derive new regular languages and relations by applying regular operators, instead of re-writing the grammars.

3.3 FINITE-STATE AUTOMATA

In our childhood, each of us must have played some game that fits the following description:

Pieces are set up on a playing board; dice are thrown or a wheel is spun, and a number is generated at random. Based on the number appearing on the dice, the pieces on the board are rearranged specified by the rules of the game. Then, it is your opponent's turn; she also does the same thing, resulting in rearrangement of the pieces based on the number generated. There is no skill or choice involved. The entire game is based on the values of the random numbers.

Consider all possible positions of the pieces on the board and call them *states*. The state in which the game begins is termed the *initial state*, and the state corresponding to the winning positions is termed the *final state*. We begin with the initial state of the starting positions of the pieces on the board. The game then changes from one state to another based on the value of the random number. For each possible number, there is one and only one resulting state given the input of the number and the current state. This continues until one player wins and the game is over. This is called a final state.

Now consider a very simple machine with an input device, a processor, some memory, and an output device. The machine starts in the initial state. It checks the input and goes to next state, which is completely determined by the prior state and the input. If all goes well, the machine

reaches final state and terminates. If the machine gets an input for which the next state is not specified, and it gets stuck at a non-final state, we say the machine has failed or rejected the input.

A general model of this type of machine is called a finite automaton; 'finite' because the number of states and the alphabet of input symbols is finite; 'automaton' because the machine moves automatically, i.e., the change of state is completely governed by the input. This type of machine is more commonly called deterministic.

A finite automaton has the following properties:

1. A finite set of states, one of which is designated the *initial* or *start state*, and one or more of which are designated as the *final states*.
2. A finite alphabet set, Σ, consisting of input symbols.
3. A finite set of *transitions* that specify for *each* state and *each* symbol of the input alphabet, the state to which it next goes.

A finite automaton can be deterministic or non-deterministic. In a non-deterministic automaton, more than one transition out of a state is possible for the same input symbol.

Example 3.2 Suppose $\Sigma = \{a, b\}$, the set of states $= \{q_0, q_1, q_2, q_3, q_4\}$ with q_0 being the start state and q_4 the final state, we have the following rules of transition:

1. From state q_0 and with input a, go to state q_1.
2. From state q_1 and with input b, go to state q_2.
3. From state q_1 and with input c, go to state q_3.
4. From state q_2 and with input b, go to state q_4.
5. From state q_3 and with input b, go to state q_4.

This finite-state automaton is shown as a directed graph, called transition diagram, in Figure 3.1. The nodes in this diagram correspond to the states, and the arcs to transitions. The arcs are labelled with inputs. The final state is represented by a double circle. As seen in the figure, there is exactly one transition leading out of each state. Hence, this automaton is deterministic.

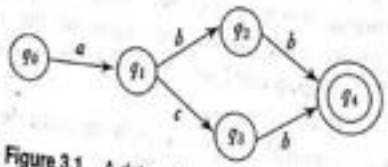


Figure 3.1 A deterministic finite-state automaton (DFA)

Finite-state automata have been used in a wide variety of areas, including linguistics, electrical engineering, computer science, mathematics, and logic. These are an important tool in computational linguistics and have been used as a mathematical device to implement regular expressions. Any regular expression can be represented by a finite automaton and the language of any finite automaton can be described by a regular expression. Both have the same expressive power. The following formal definitions of the two types of finite state automaton, namely, deterministic and non-deterministic finite automaton, are taken from Hopcroft and Ullman (1979).

A *deterministic finite-state automaton* (DFA) is defined as a 5-tuple $(Q, \Sigma, \delta, S, F)$, where Q is a set of *states*, Σ is an alphabet, S is the *start* state, $F \subseteq Q$ is a set of *final* states, and δ is a *transition function*. The transition function δ defines mapping from $Q \times \Sigma$ to Q. That is, for each state q and symbol a, there is at most one transition possible as shown in Figure 3.1.

Unlike DFA, the transition function of a *non-deterministic finite-state automaton* (NFA) maps $Q \times (\Sigma \cup \{\varepsilon\})$ to a subset of the power set of Q. That is, for each state, there can be more than one transition on a given symbol, each leading to a different state.

This is shown in Figure 3.2, where there are two possible transitions from state q_0 on input symbol a.

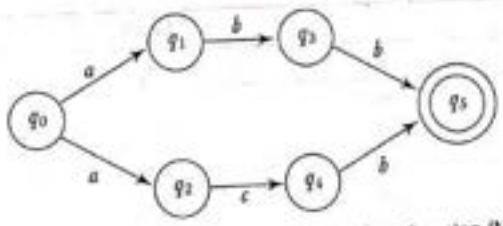


Figure 3.2 Non-deterministic finite-state automaton (NFA)

A *path* is a sequence of transitions beginning with the start state. A path leading to one of the final states is a *successful path*. The FSAs encode *regular* languages. The language that an FSA encodes is the set of strings that can be formed by concatenating the symbols along each successful path. Clearly, for automata with cycles, these sets are not finite.

We now examine what happens to various input strings that are presented to finite state automata. Consider the deterministic automaton described in Example 3.2 and the input, *ac*. We start with state q_0 and go to state q_1. The next input symbol is c, so we go to state q_3. No more input is left and we have not reached the final state, i.e., we have an unsuccessful end. Hence, the string *ac* is not recognized by the automaton.

This example illustrates how an FSA can be used to accept or recognize a string. The set of all strings that leave us in a final state is called the language accepted or defined by the FA. This means *ac* is not a word in the language defined by the automaton of Figure 3.1.

Now, consider the input acb. Again, we start with state q_0 and go to state q_1. The next input symbol is *c*, so we go to state q_3. The next input symbol is *b*, which leads to state q_4. No more input is left and we have reached to final state, i.e., this time we have a successful termination. Hence, the string *acb* is a word of the language defined by the automaton.

The language defined by this automaton can be described by the regular expression */abb|acb/*.

The example considered here is quite simple. Typically, the list of transition rules can be quite long. Listing all transition rules may be inconvenient, so often we represent an automaton as a *state-transition table*. The rows in this table represent states and the columns correspond to input. The entries in the table represent the transition corresponding to a given state-input pair. A ϕ entry indicates missing transition. This table contains all the information needed by FSA. The state transition table for the automaton considered in Example 3.2 is shown in Table 3.5.

Table 3.5 The state-transition table for the DFA shown in Figure 3.1

State	Input		
	a	*b*	*c*
start: q_0	q_1	ϕ	ϕ
q_1	ϕ	q_2	q_3
q_2	ϕ	q_4	ϕ
q_3	ϕ	q_4	ϕ
final: q_4	ϕ	ϕ	ϕ

Now, consider a language consisting of all strings containing only *a*s and *b*s and ending with *baa*. We can specify this language by the regular expression */(a|b)*baa$/*. The NFA implementing this regular expression is shown in Figure 3.3.

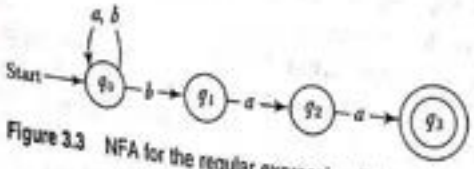


Figure 3.3 NFA for the regular expression */(a|b)*baa$/*

Table 3.6 The state-transition table for the NFA shown in Figure 3.3.

State	Input		
	a	*b*
start: q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_2\}$	ϕ
q_2	$\{q_3\}$	ϕ
final: q_3	ϕ	ϕ

Two automata that define the same language are said to be *equivalent*. An NFA can be converted to an equivalent DFA and vice versa. The equivalent DFA for the NFA shown in Figure 3.3 is shown in Figure 3.4.

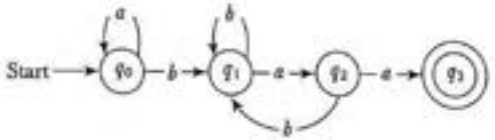


Figure 3.4 Equivalent DFA for NFA in Figure 3.3

3.4 MORPHOLOGICAL PARSING

Morphology is a sub-discipline of linguistics. It studies word structure and the formation of words from smaller units (morphemes). The goal of morphological parsing is to discover the morphemes that build a given word. Morphemes are the smallest meaning-bearing units in a language. For example, the word 'bread' consists of a single morpheme and 'eggs' consist of two: the morpheme egg and the morpheme -s. A morphological parser should be able to tell us that the word 'eggs' is the plural form of the noun stem 'egg'.

There are two broad classes of morphemes: stems and affixes. The stem is the main morpheme, i.e., the morpheme that contains the central meaning. Affixes modify the meaning given by the stem. Affixes are divided into prefix, suffix, infix, and circumfix. Prefixes are morphemes which appear before a stem, and suffixes are morphemes applied to the end of the stem. Circumfixes are morphemes that may be applied to either end of the stem while infixes are morphemes that appear inside a stem. Prefixes and suffixes are quite common in Urdu, Hindi, and English. For example, the Urdu word, بےوقت (*bewaqt*), meaning untimely, is composed of the stem, *waqt*, and the prefix, *be-*, گھوڑوں (*ghodhon*) is composed of the stem, گھوڑا (*ghodha*) and the suffix, وں (*on*). Also, the word, ضرورت مند (*zarooratmand*), is composed of the stem, ضرورت

(*zaroorat*), and the suffix, مند (*mand*). Similarly, the English word, unhappy, is composed of the stem, happy, and the prefix, un-. The English word, birds, is composed of the stem, bird, and the suffix, -s. Likewise, the Hindi word, शीतला is composed of a stem शीतल and the suffix – आ.

Telgu word గుర్రములు (*gurramulu*–plural form of *gurramu*, meaning horse) is composed of the stem గుర్రము (*gurramu*) and suffix -లు (*lu*). Some commonly used suffixes in plural forms of Telugu nouns are: లు (*lu*), ళ్లు (*llu*), ళ్లు (*dlu*). Here is a list of singular and plural forms of a few Telugu words:

Singular	Meaning	Plural
పిల్లి (*pilli*)	Cat	పిల్లులు (*pillulu*)
పడుచు (*paduchu*)	Women	పడుచులు (*paduchulu*)
ఆడది (*aadadi*)	Women	ఆడవాండ్లు (*aadawandlu** or *aadawalu*)
మగవాడు (*magavadu*)	Man	మగవాండ్లు (*magavandlu* or *magavallu*)
పురుషుడు (*purushudu*)	Man	పురుషులు (*purushulu*)
చెవి (*chevi*)	Ear	చెవులు (*chevulu*)
ఇల్లు (*illu*)	House	ఇల్లులు (*illulu* or *illlu*)

*Another plural form of *aadadi* is *adawaru*, which is used to show respect.

There are three main ways of word formation: inflection, derivation, and compounding. In inflection, a root word is combined with a grammatical morpheme to yield a word of the same class as the original stem. Derivation combines a word stem with a grammatical morpheme to yield a word belonging to a different class, e.g., formation of the noun 'computation' from the verb 'compute'. The formation of a noun from a verb or adjective is called nominalization. Compounding is the process of merging two or more words to form a new word. For example, personal computer, desktop, overlook. Morphological analysis and generation deal with inflection, derivation and compounding process in word formation.

New words are continually forming a natural language. Many of these are morphologically related to known words. Understanding morphology is therefore important to understand the syntactic and semantic properties of new words. Morphological analysis and generation are essential to many NLP applications ranging from spelling corrections to machine translations. In parsing, e.g., it helps to know the agreement features of words. In information retrieval, morphological analysis helps identify the presence of a query word in a document in spite of different morphological variants.

Parsing, in general, means taking an input and producing some sort of structures for it. In NLP, this structure might be morphological, syntactic, semantic, or pragmatic. Morphological parsing takes as input the inflected

surface form of each word in a text. As output, it produces the parsed form consisting of a canonical form (or *lemma*) of the word and a set of tags showing its syntactical category and morphological characteristics, e.g., possible part of speech and/or inflectional properties (gender, number, person, tense, etc.). Morphological generation is the inverse of this process. Both analysis and generation rely on two sources of information: a dictionary of the valid lemmas of the language and a set of inflection paradigms.

A morphological parser uses following information sources:

1. **Lexicon**
   A lexicon lists stems and affixes together with basic information about them.
2. **Morphotactics**
   There exists certain ordering among the morphemes that constitute a word. They cannot be arranged arbitrarily. For example, rest-less-ness is a valid word in English but not rest-ness-less. Morphotactics deals with the ordering of morphemes. It describes the way morphemes are arranged or touch each other.
3. **Orthographic rules**
   These are spelling rules that specify the changes that occur when two given morphemes combine. For example the $y \rightarrow ier$ spelling rule changes '*easy*' to '*easier*' and not to '*easyer*'.

Morphological analysis can be avoided if an exhaustive lexicon is available that lists features for all the word-forms of all the roots. Given a word, we simply consult the lexicon to get its feature values. For example, suppose an exhaustive lexicon for Hindi contains the following entries for the Hindi root-word *ghodhaa*.

Table 3.7 A sample lexicon entry

Word form	Category	Root	Gender	Number	Person
Ghodhaa	noun	GhoDaa	masculine	singular	3rd
Ghodhii	-do-	-do-	feminine	-do-	-do-
Ghodhon	-do-	-do-	masculine	plural	-do-
Ghodhe	-do-	-do-	-do-	-do-	-do-

Given a word, say *ghodhon*, we can look up the lexicon to get its feature values.

However, this approach has several limitations. First, it puts a heavy demand on memory. We have to list every form of the word, which results in a large number of, often redundant, entries in the lexicon.

Second, an exhaustive lexicon fails to show the relationship between different roots having similar word-forms. That means the approach fails to capture linguistic generalization, which is essential to develop a system capable of understanding unknown words.

Third, for morphologically complex languages, like Turkish, the number of possible word-forms may be theoretically infinite. It is not practical to list all possible word-forms in these languages.

These limitations explain why morphological parsing is necessary. The complexity of the morphological analysis varies widely among the world's languages, and is quite high even in relatively simple cases, such as English.

The simplest morphological systems are stemmers that collapse morphological variations of a given word (word-forms) to one lemma or stem. They do not require a lexicon. Stemmers have been especially used in information retrieval. Two widely used stemming algorithms have been developed by Lovins (1968) and Porter (1980). Stemmers do not use a lexicon; instead, they make use of rewrite rules of the form:

ier → *y* (e.g., earlier → early)

ing → ε (e.g., playing → play)

Stemming algorithms work in two steps:

(i) Suffix removal: This step removes predefined endings from words.

(ii) Recoding: This step adds predefined endings to the output of the first step.

These two steps can be performed sequentially as in Lovins's stemmer or simultaneously as in Porter's stemmer. For example, Porter's stemmer makes use of the following transformation rule:

ational → *ate*

to transform word such as 'rotational' into 'rotate'.

It is difficult to use stemming with morphologically rich languages. Even in English, stemmers are not perfect. Krovitz (1993) pointed out errors of omissions and commissions in the Porter algorithm, such as transformation of the word 'organization' into 'organ' and 'noise' into 'noisy'. Another problem with Porter's algorithm is that it reduces only suffixes; prefixes and compounds are not reduced.

A more efficient *two-level morphological model*, first proposed by Koskenniemi (1983), can be used for highly inflected languages. In this model, a word is represented as a correspondence between its lexical level form and its surface level form. The surface level represents the actual spelling of the word while the lexical level represents the concatenation of its constituent morphemes. Morphological parsing is viewed as a mapping from the surface level into morpheme and feature sequences on the lexical level.

For example, the surface form 'playing' is represented in the lexical form as play + V + PP as shown in Figure 3.5. The lexical form consists of the stem 'play' followed by the morphological information +V +PP, which tells us that 'playing' is the present participle form of the verb.

Surface level	p	l	a	y	i	n	g
Lexical level	p	l	a	y	+V	PP	

Figure 3.5 Surface and lexical forms of a word

Similarly, the surface form 'books' is represented in the lexical form as 'book + N + PL', where the first component is the stem and the second component (N + PL) is the morphological information, which tells us that the surface level form is a plural noun.

This model is usually implemented with a kind of finite-state automata, called *finite-state transducer* (FST). A transducer maps a set of symbols to another. A finite state transducer does this through a finite state automaton. An FST can be thought of as a two-state automaton, which recognizes or generates a pair of strings. FST passes over the input string by consuming the input symbols on the tape it traverses and consists it to the output string in the form of symbols. Formally, an FST has been defined by Hopcroft and Ullman (1979) as follows:

A finite-state transducer is a 6-tuple $(\Sigma_1, \Sigma_2, Q, \delta, \bar{S}, F)$, where θ is set of states, S is the initial state, and $F \subseteq Q$ is a set of final states, Σ_1 is *input* alphabet, Σ_2 is *output* alphabet, and δ is a function mapping $Q \times (\Sigma_1 \cup \{\varepsilon\}) \times (\Sigma_2 \cup \{\varepsilon\})$ to a subset of the power set of Q.

Transducers can be seen as automata with transitions labelled with symbols from $\Sigma_1 \times \Sigma_2$, where Σ_1 and Σ_2 are the alphabets of input and output respectively. Thus, an FST is similar to an NFA except in that transitions are made on strings rather than on symbols and, in addition, they have outputs.

Figure 3.6 shows a simple transducer that accepts two input strings, hot and cat, and maps them onto cot and bat respectively. It is a common practice to represent a pair like *a:a* by a single letter.

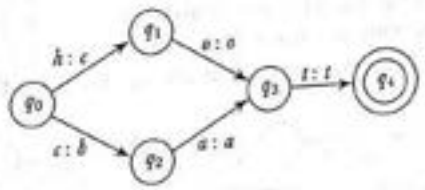


Figure 3.6 Finite-state transducer

Just as FSAs encode regular languages, FSTs encode regular relations. Regular relation is the relation between regular languages. The regular language encoded on the upper side of an FST is called upper language, and the one on the lower side is termed lower language. If T is a transducer, and s is a string, then we use $T(s)$ to represent the set of strings encoded by T such that the pair (s, t) is in the relation.

The FSTs are closed under union concatenation, composition, and Kleene closure. However, in general, they are not closed under intersection and complementation.

With this introduction, we can now implement the two-level morphology using FST. To get from the surface form of a word to its morphological analysis, we proceed in two steps as illustrated in Figure 3.7.

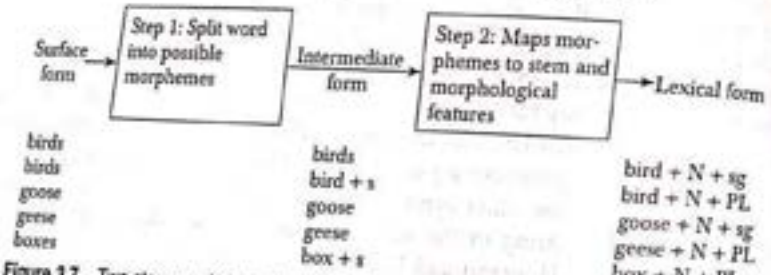


Figure 3.7 Two-step morphological parser

First, we split the words up into its possible components. For example, we make *bird* + *s* out of *birds*, where + indicates morpheme boundaries. In this step, we also consider spelling rules. Thus, there are two possible ways of splitting up *boxes*, namely *boxe* + *s* and *box* + *s*. The first one assumes that *boxe* is the stem and *s* the suffix, while the second assumes that *box* the stem is and that *e* has been introduced due to the spelling rule. The output of this step is a concatenation of morphemes, i.e., stems and affixes. There can be more than one representation for a given word. A transducer that does the mapping (translation) required by this step for the surface form 'lesser' might look like Figure 3.8. This FST represents the information that the comparative form of the adjective *less* is *lesser*; ε here is the empty string. The automaton is inherently bi-directional: the same transducer can be used for analysis (surface input, 'upward' application) or for generation (lexical input, 'downward' application).

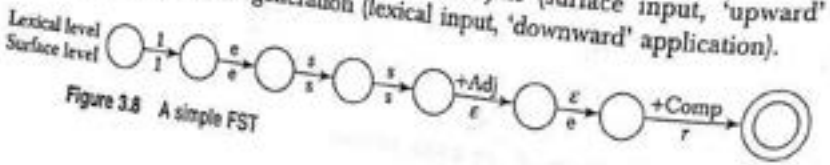


Figure 3.8 A simple FST

In the second step, we use a lexicon to look up categories of the stems and meaning of the affixes. So, *bird* + *s* will be mapped to *bird*+ *N*+ *PL*, and *box* + *s* to *box*+ *N* + *PL*. We also find out now that *boxe* is not a legal stem. This tells us that splitting *boxes* into *boxe* + *s* is an incorrect way of splitting *boxes*, and should therefore be discarded. This may not be the case always. We have words like *spouses* or *parses* where splitting the word into *spouse* + *s* or *parse* + *s* is correct. Orthographic rules are used to handle these spelling variations. For instance, one of the spelling rules says- add e after -s, -z, -x, -ch, -sh before the s (e.g., dish→dishes, box→boxes).

Each of these steps can be implemented with the help of a transducer. Thus, we need to build two transducers: one that maps the surface form to the intermediate form and another that maps the intermediate form to the lexical form.

We now develop an FST-based morphological parser for singular and plural nouns in English. The plural form of regular nouns usually end with -s or -es. However, a word ending in 's' need not necessarily be the plural form of a word. There are a number of singular words ending in s, e.g., miss and ass. One of the required translations is the deletion of the 'e' when introducing a morpheme boundary. This deletion is usually required for words ending in xes, ses, zes (e.g., suffixes and boxes). The transducer in Figure 3.9 does this. Figure 3.10 shows the possible sequences of states that the transducer undergoes, given the surface forms *birds* and *boxes* as input.

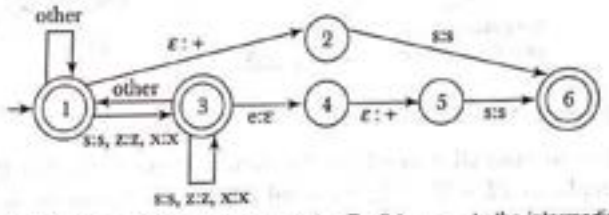


Figure 3.9 A simplified FST, mapping English nouns to the intermediate form

The next step is to develop a transducer that does the mapping from the intermediate level to the lexical level. The input to transducer has one of the following forms:

- Regular noun stem, e.g., *bird, cat*
- Regular noun stem + s, e.g., *bird* + *s*
- Singular irregular noun stem, e.g., *goose*
- Plural irregular noun stem, e.g., *geese*

In the first case, the transducer has to map all symbols of the stem to themselves and then output *N* and *sg* (Figure 3.7). In the second case, it

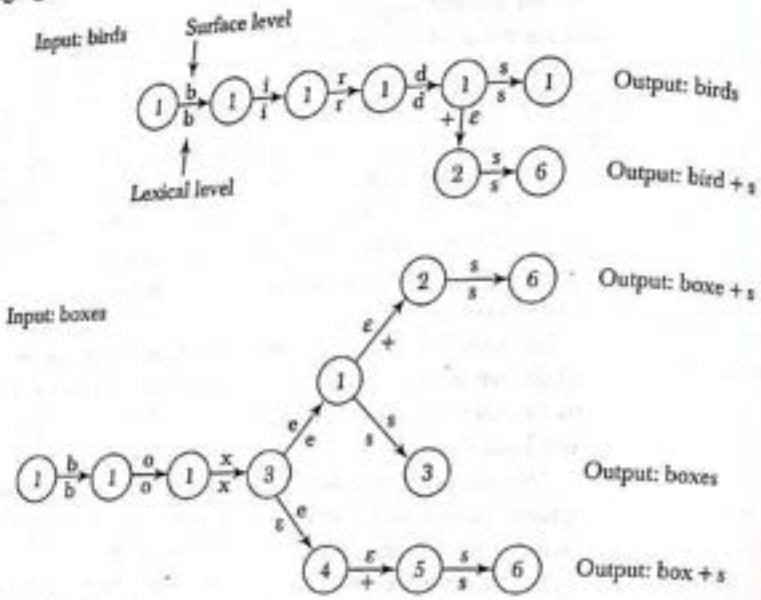


Figure 3.10 Possible sequences of states

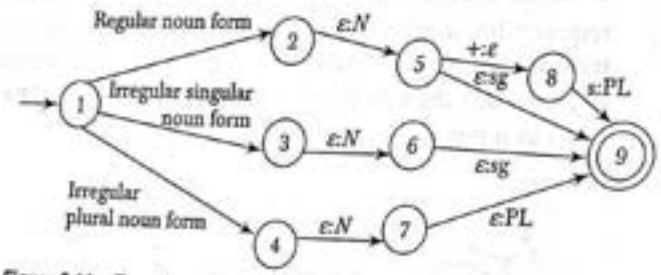


Figure 3.11 Transducer for Step 2

has to map all symbols of the stem to themselves, but then output *N* and replaces *PL* with *s*. In the third case, it has to do the same as in the first case. Finally, in the fourth case, the transducer has to map the irregular plural noun stem to the corresponding singular stem (e.g., *geese* to *goose*) and then it should add *N* and *PL*. The general structure of this transducer looks like Figure 3.11.

The mapping from State 1 to State 2, 3, or 4 is carried out with the help of a transducer encoding a lexicon. The transducer implementing the lexicon maps the individual regular and irregular noun stems to their correct noun stem, replacing labels like regular noun form, etc. This lexicon maps the surface form *geese*, which is an irregular noun, to its correct stem *goose* in the following way:

g:g e:o e:o s:s e:e

Mapping for the regular surface form of *bird* is b:b i:i r:r d:d. Representing pairs like *a:a* with a single letter, these two representations are reduced to g e:o e:o s e and b i r d respectively.

Composing this transducer with the previous one, we get a single two-level transducer with one input tape and one output tape. This maps plural nouns into the stem plus the morphological marker + pl and singular nouns into the stem plus the morpheme + sg. Thus a surface word form *birds* will be mapped to bird + N + pl as follows.

b:b i:i r:r d:d + ε:N + s:pl

Each letter maps to itself, while ε maps to morphological feature +N, and s maps to morphological feature pl. Figure 3.12 shows the resulting composed transducer.

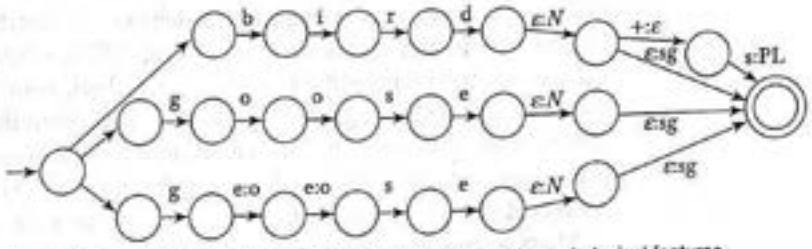


Figure 3.12 A transducer mapping nouns to their stem and morphological features

The power of the transducer lies in the fact that the same transducer can be used for analysis and generation. That is, we can run it in the downward direction (input: surface form and output: lexical form) or in the upward direction.

3.5 SPELLING ERROR DETECTION AND CORRECTION

In computer-based information systems, errors of typing and spelling constitute a very common source of variation between strings. These errors have been widely investigated. All investigations agree that single character omission, insertion, substitution, and reversal are the most common typing mistakes. In an early investigation, Damearu (1964) reported that over 80% of the typing errors were single-error misspellings:

(1) substitution of a single letter,
(2) omission of a single letter,
(3) insertion of a single letter, and
(4) transposition of two adjacent letters.

Shafer and Hardwick (1968) found that the most common type of single character error was substitution, followed by omission of a letter and then insertion of a letter. Single character omission occurs when a single character is missed (deleted), e.g., when 'concept' is accidentally typed as 'concpt'. Insertion error refers to the presence of an extra character in a word, e.g. when 'error' is misspell as 'errorn'. Substitution error occurs when a wrong letter is typed in place of the right one, as in 'errpr', where 'p' appears in place of 'o'. Reversal refers to a situation in which the sequence of characters is reversed, e.g., 'aer' instead of 'are'. This is also termed transposition.

Optical character recognition (OCR) and other automatic reading devices introduce errors of substitution, deletion, and insertion but not of reversal. OCR errors are usually grouped into five classes: substitution, multi-substitution (or framing), space deletion or insertion, and failures. Unlike substitution errors made in typing, OCR substitution errors are caused due to visual similarity such as c→e, 1→l, r→n. The same is true for multi-substitution, e.g., m→rn. Failure occurs when the OCR algorithm fails to select a letter with sufficient accuracy. The frequency and type of errors are characteristics of the particular device. These errors can be corrected using 'context' or by using linguistic structures.

Many approaches to speech recognition deal with strings of phonemes (or symbols representing sounds), and attempt to match a spoken utterance with a dictionary of known utterances.

Unlike typing errors, spelling errors are mainly phonetic, where the misspell word is pronounced in the same way as the correct word. Phonetic errors are harder to set right because they distort the word by more than a single insertion, deletion, or substitution. Phonetic variations are common in transliteration. For example,

Spelling errors belong to one of two distinct categories: non-word errors and real word errors. When an error results in a word that does not appear in a given lexicon or is not a valid orthographic word form, it is termed a non-word error. Most of the early research on spelling errors focused on the detection of such non-words. The two main techniques used were *n*-gram analysis and dictionary lookup. Non-word error detection is now considered a solved problem.

A real-word error results in actual words of the language. It occurs due to typographical mistakes or spelling errors, e.g., substituting the spelling of a homophone or near-homophone, such as piece for peace or meat for meet. Real-word errors may cause local syntactic errors, global syntactic

errors, semantic errors, or errors at discourse or pragmatic levels. It becomes impossible to decide that a word is wrong without some contextual information.

Spelling correction consists of *detecting* and *correcting* errors. Error detection is the process of finding misspelled words and error correction is the process of suggesting correct words to a misspelled one. These sub-problems are addressed in two ways:

1. Isolated-error detection and correction
2. Context-dependent error detection and correction

In isolated-word error detection and correction, each word is checked separately, independent of its context. Detecting whether or not a word is correct seems simple–why not to look up the word in a lexicon? Unfortunately, it is not as simple as it appears. There are a number of problems associated with this simple strategy.

- The strategy requires the existence of a lexicon containing all correct words. Such a lexicon would take a long time to compile and occupy a lot of space.
- Some languages are highly productive. It is impossible to list all the correct words of such languages.
- This strategy fails when spelling error produces a word that belongs to the lexicon, e.g., when 'theses' is written in place of 'these'. Such an error is called a *real-word error*.
- The larger the lexicon, the more likely it is that an error goes undetected, because the chance of a word being found is greater in a large lexicon.

Context dependent error detection and correction methods, utilize the context of a word to detect and correct errors. This requires grammatical analysis and is thus more complex and language dependent. Even in context dependent methods, the list of candidate words must first be obtained using an isolated-word method before making a selection depending on the context.

The spelling correction algorithm has been broadly categorized by Kukich (1992) as follows.

Minimum edit distance The minimum edit distance between two strings is the minimum number of operations (insertions, deletions, or substitutions) required to transform one string into another. Spelling correction algorithms based on minimum edit distance are the most studied algorithms.

Similarity key techniques The basic idea in a similarity key technique is to change a given string into a key such that similar strings will change into the same key. The SOUNDEX system (Odell and Russell 1918) is an example of a system that uses this technique in phonetic spelling correction applications.

***n*-gram based techniques** The *n*-grams can be used for both non-word and real-word error detection because in the English alphabet, certain bi-grams and tri-grams of letters never occur or rarely do so; for example the tri-gram *qst* and the bi-gram *qd*. This information can be used to handle non-word error. Strings that contain these unusual *n*-grams can be identified as possible spelling errors. *n*-gram techniques usually require a large corpus or dictionary as training data, so that an *n*-gram table of possible combinations of letters can be compiled. In case of real-word error detection, we calculate the likelihood of one character following another and use this information to find possible correct word candidates.

Neural nets These have the ability to do associative recall based on incomplete and noisy data. They can be trained to adapt to specific spelling error patterns. The drawback of neural nets is that they are computationally expensive.

Rule-based techniques In a rule-based technique, a set of rules (heuristics) derived from knowledge of a common spelling error pattern is used to transform misspelled words into valid words. For example, if it is known that many errors occur from the letters *ie* being typed as *ei*, then we may write a rule that represents this.

3.5.1 Minimum Edit Distance

The minimum edit distance is the number of insertions, deletions, and substitutions required to change one string into another (Wagner and Fischer 1974). When we talk about distance between two strings, we are talking of the minimum edit distance. For example, the minimum edit distance between 'tutor' and 'tumor' is 2: We substitute 'm' for 't' and insert 'u' before 'r'. No smaller edit sequence can be found for this conversion. Therefore, the minimum edit distance is 2. Edit distance between two strings can be represented as a binary function, ed, which maps two strings to their edit distance. ed is symmetric. For any two strings, *s* and *t*, ed(*s*, *t*) is always equal to ed(*t*, *s*).

Edit distance can be viewed as a string alignment problem. By aligning two strings, we can measure the degree to which they match. There may be more than one possible alignment between two strings. The best

possible alignment corresponds to the minimum edit distance between the strings. The alignment shown here, between *tutor* and *tumour*, has a distance of 2.

```
t  u  t  o  –  r
t  u  m  o  u  r
```

A dash in the upper string indicates insertion. A substitution occurs when the two alignment symbols do not match (shown in bold). We can associate a weight or cost with each operation. The *Levenshtein* distance between two sequences is obtained by assigning a unit cost to each operation. Another possible alignment for this sequences is:

```
t  u  t  –  o  –  r
t  u  –  m  o  u  r
```

which has a cost of 3. We already have a better alignment than this one.

The problem of finding minimum edit distance seems quite simple but in fact is not so. A choice that seems good initially might lead to problems later. Dynamic programming algorithms can be quite useful for finding minimum edit distance between two sequences. Dynamic programming refers to a class of algorithms that apply a table-driven approach to solve problems by combining solutions to sub-problems. The dynamic programming algorithm for minimum edit distance is implemented by creating an edit distance matrix. This matrix has one row for each symbol in the source string and one column for each matrix in the target string. The (i, j)th cell in this matrix represents the distance between the first i character of the source and the first j character of the target string. Each cell can be computed as a simple function of its surrounding cells. Thus, by starting at the beginning of the matrix, it is possible to fill each entry iteratively. The value in each cell is computed in terms of three possible paths.

$$dist[i, j] = \begin{cases} dist[i-1, j] + insert_cost, \\ dist[i-1, j-1] + subst_cost[source_i, target_j] \\ dist[i, j-1] + delete_cost \end{cases}$$

The substitution will be 0 if the ith character in the source matches with jth character in the target. The minimum edit distance algorithm is shown in Figure 3.13. How the algorithm computes the minimum edit distance between *tutor* and *tumour* is shown in Figure 3.14.

```
Input: Two strings, X and Y
Output: The minimum edit distance between X and Y
m ← length(X)
n ← length(Y)
for i = 0 to m do
    dist[i,0] ← i
for j = 0 to n do
    dist[0,j] ← j
for i = 0 to m do
    for j = 0 to n do
        dist[i,j] = min{ dist[i-1,j] + insert_cost,
                         dist[i-1,j-1] + subst_cost(X_i, Y_j),
                         dist[i,j-1] + delet_cost }
```

Figure 3.13 Minimum edit distance algorithm

	#	t	u	m	o	u	r
#	0	1	2	3	4	5	6
t	1	0	1	2	3	4	5
u	2	1	0	1	2	3	4
t	3	2	1	1	2	3	4
o	4	3	2	2	1	2	3
r	5	4	3	3	2	2	2

Figure 3.14 Computing minimum edit distance

Minimum edit distance algorithms are also useful for determining accuracy in speech recognition systems. Kemal Oflazer (1996) proposed an efficient algorithm based on spelling correction with finite-state automata.

3.6 WORDS AND WORD CLASSES

Words are classified into categories called part-of-speech. These are sometimes called word classes or lexical categories. These lexical categories are usually defined by their syntactic and morphological behaviours. The most common lexical categories are nouns and verbs. Other lexical categories include adjectives, adverbs, prepositions, and conjunctions. Table 3.8 shows some of the word classes in English. Lexical categories and their properties vary from language to language. Word classes are further categorized as open and closed word classes. Open word classes constantly

acquire new members while closed word classes do not (or only infrequently do so). Nouns, verbs (except auxiliary verbs), adjectives, adverbs, and interjections are open word classes. Prepositions, auxiliary verbs, delimiters, conjunction, and particles are closed word classes.

Table 3.8 Part-of-speech example

NN	noun	student, chair, proof, mechanism
VB	verb	study, increase, produce
ADJ	adj	large, high, tall, few,
JJ	adverb	carefully, slowly, uniformly
IN	preposition	in, on, to, of
PRP	pronoun	I, me, they
DET	determiner	the, a, an, this, those

3.7 PART-OF-SPEECH TAGGING

Part-of-speech tagging is the process of assigning a part-of-speech (such as a noun, verb, pronoun, preposition, adverb, and adjective), to each word in a sentence. The input to a tagging algorithm is the sequence of words of a natural language sentence and specified tag sets (a finite list of part-of-speech tags). The output is a single best part-of-speech tag for each word. Many words may belong to more than one lexical category. For example, the English word 'book' can be a noun as in '*I am reading a good book*' or a verb as in '*The police booked the snatcher*'. The same is true for other languages. For example, the Hindi word '*sona*' may mean 'gold' (noun) or 'sleep' (verb). However, only one of the possible meanings is used at a time. In tagging, we try to determine the correct lexical category of a word in its context. No tagger is efficient enough to identify the correct lexical category of each word in a sentence in every case. The tag assigned by a tagger is the most likely for a particular use of word in a sentence.

The collection of tags used by a particular tagger is called a *tag set*. Most part-of-speech tag sets make use of the same basic categories, i.e., noun, verb, adjective, and prepositions. However, tag sets differ in how they define categories and how finely they divide words into categories. For example, both *eat* and *eats* might be tagged as a verb in one tag set, but assigned distinct tags in another tag set. In addition, most tag sets capture morpho-syntactic information such as singular/plural, number, gender, tense, etc. Consider the following sentences:

Zuha eats an apple daily.

Aman ate an apple yesterday.

They have eaten all the apples in the basket.

I like to eat guavas.

The word *eat* has a distinct grammatical form in each of these four sentences. *Eat* is the base form, *ate* its past tense, and the form *eats* requires a third person singular subject. Similarly, *eaten* is the past participle form and cannot occur in another grammatical context. It is required after have or has. Thus, the following sentences are ungrammatical:

I like to eats guava.

They eaten all the apples.

The number of tags used by different taggers varies substantially. Some use 20, while others use over 400 tags. The Penn Treebank tag set contains 45 tags while C7 uses 164. For a language like English, which is not morphologically rich, the C7 tagset is too big. The tagging process would yield too many mistagged words and the result would have to be manually corrected. Despite this, bigger tag sets have been used, e.g., TOSCA-ICE for the International Corpus of English with 270 tags (Garside 1997), or TESS with 200 tags. The larger the tag set, the greater the information captured about a linguistic context. However, the task of tagging becomes complicated and requires manual correction. A bigger tag set can be used for morphologically rich languages without introducing too many tagging errors. A tag set that uses just one tag to denote all the verbs will assign identical tags to all the forms of a verb. Although this coarse-grained distinction may be appropriate for some tasks, a fine-grained tag set captures more information. This is useful for tasks like syntactic pattern detection. The Penn Treebank tag set captures finer distinctions by assigning distinct tags to distinct grammatical forms of a verb, as summarized in Table 3.9. Tags assigned to the four different forms of the word *eat* according to this tag set is shown in Table 3.10.

Table 3.9 Tags from Penn Treebank tag set

VB	Verb, base form Subsumes imperatives, infinitives, and subjunctives
VBD	Verb, past tense Includes the conditional form of the verb *to be*
VBG	Verb, gerund, or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present

Table 3.10 Possible tags for the word to *eat*

eat	VB
ate	VBD
eaten	VBN
eats	VBP

Here is an example of a tagged sentence:

Speech/NN sounds/NNS were/VBD sampled/VBN by/IN a/DT microphone/NN.

The tag set used is Penn Treebank.

Another tagging possible for this sentence is as follows:

Speech/NN sounds/VBZ were/VBD sampled/VBN by/IN a/DT microphone/NN.

It is easy to see that the second tagged sequence is not correct. It leads to semantic incoherence. We resolve the ambiguity using the context of the word. The context is also utilized by automatic taggers.

Part-of-speech tagging is an early stage of text processing in many NLP applications including speech synthesis, machine translation, information retrieval, and information extraction. In information retrieval, part-of-speech tagging can be used for indexing (for identifying useful tokens like nouns and phrases) and for disambiguating word senses. Tagging is not as complex as parsing. In tagging, a complete parse tree is not built; part-of-speech is assigned to words using contextual information.

Part-of-speech tagging methods fall under the three general categories.

- Rule-based (linguistic)
- Stochastic (data-driven)
- Hybrid

Rule-based taggers use hand-coded rules to assign tags to words. These rules use a lexicon to obtain a list of candidate tags and then use rules to discard incorrect tags.

Stochastic taggers have data-driven approaches in which frequency-based information is automatically derived from corpus and used to tag words. Stochastic taggers disambiguate words based on the probability that a word occurs with a particular tag. The simplest scheme is to assign the most frequent tag to each word. An early example of stochastic tagger was CLAWS (constituent likelihood automatic word-tagging system). CLAWS is the Stochastic equivalent of TAGGIT. Hidden Markov model (HMM) is tshe standard Stochastic tagger.

Hybrid taggers combine features of both these approaches. Like rule-based systems, they use rules to specify tags. Like stochastic systems, they use machine-learning to induce rules from a tagged training corpus automatically. The transformation-based tagger or Brill tagger is an example of the hybrid approach.

3.7.1 Rule-based Tagger

Most rule-based taggers have a two-stage architecture. The first stage is simply a dictionary look-up procedure, which returns a set of potential tags (parts-of-speech) and appropriate syntactic features for each word. The second stage uses a set of hand-coded rules to discard contextually illegitimate tags to get a single part-of-speech for each word. For example, consider the noun-verb ambiguity in the following sentence:

The show must go on.

The potential tags for the word *show* in this sentence is {VB, NN}. We resolve this ambiguity by using the following rule.

IF *preceding word is determiner* THEN *eliminate VB tag.*

This rule simply disallows verbs after a determiner. Using this rule the word *show* in the given sentence can only be noun.

In addition to contextual information, many taggers use morphological information to help in the disambiguation process. An example of a rule that make use of morphological information is:

IF *word ends in –ing and preceding word is a verb* THEN *label it a verb (VB).*

Capitalization information can be utilized in the tagging of unknown nouns. Rule-based taggers usually require supervised training. Instead, rules can be induced automatically. To induce rules untagged text is run through a tagger. The output is then manually corrected. The corrected text is then submitted to the tagger, which learns correction rules by comparing the two sets of data. This process may be repeated several times.

The earlier systems for automatic tagging were all rule-based. An example is TAGGIT (Greene and Rubin 1971), which was used for the initial tagging of the Brown corpus (Francis and Kucera 1982). This was also rule-based system. It used 3,300 disambiguation rules and was able to tag 77% of the words in the Brown corpus with their correct part-of-speech. The rest was done manually over several years. Yet another rule-based tagger is ENGTWOL (Voutilainen 1995).

Speed is an advantage of the rule-based tagger, and unlike stochastic taggers, they are deterministic. One of the arguments against them is the skill and effort required in writing disambiguation rules. However, stochastic taggers also require manual work if good performance is to be achieved. In the rule-based system, time is spent in writing a rule-set. For stochastic taggers, time is spent developing restrictions on transitions and emissions to improve tagger performance. Another disadvantage of the rule-based tagger is that it is usable for only one language. Using it for another one requires a rewrite of most of the program; unlike a probabilistic tagger which would be usable with only slight changes and new training.

3.7.2 Stochastic Tagger

The standard stochastic tagger algorithm is the HMM tagger. A Markov model applies the simplifying assumption that the probability of a chain of symbols can be approximated in terms of its parts or *n*-grams. The simplest *n*-gram model is the unigram model, which assigns the most likely tag (part-of-speech) to each token.

The unigram model needs to be trained using a tagged training corpus before it can be used to tag data. The most likely statistics are gathered over the corpus and used for tagging. The context used by the unigram tagger is the text of the word itself. For example, it will assign the tag JJ for each occurrence of *fast*, since *fast* is used as an adjective more frequently than it is used as a noun, verb, or adverb. This results in incorrect tagging in each of the following sentences:

She had a *fast*. (3.1)

Muslims *fast* during Ramadan. (3.2)

Those who were injured in the accident need to be helped *fast*. (3.3)

In the first sentence, *fast* is used as a noun. In the second, it is a verb, and in the third, an adverb.

We would expect more accurate predictions if we took more context into account when making a tagging decision. A bi-gram tagger uses the current word and the tag of the previous word in the tagging process. As the tag sequence "DT NN" is more likely than the tag sequence "DT JJ", a bi-gram model will assign a correct tag to the word *fast* in sentence (3.1). Similarly, it is more likely that an adverb (rather than a noun or an adjective) follows a verb. Hence, in sentence (3.3), the tag assigned to *fast* will be RB.

In general, an n-gram model considers the current word and the tag of the previous $n-1$ words in assigning a tag to a word. The context considered by a tri-gram model is shown in Figure 3.15. The area shaded in grey represents the context.

Tokens	w_{n-2}	w_{n-1}	w_n	w_{n+1}
tags	t_{n-2}	t_{n-1}	t_n	t_{n+1}

Figure 3.15 Context used by a tri-gram tagger

So far, we have considered how a tag is assigned to a word given the previous tag(s). However, the objective of a tagger is to assign a tag sequence to a given sentence. We now discuss how the HMM tagger assigns the most likely tag sequence to a given sentence. We call this the HMM because it uses two layers of states: a visible layer corresponding to the input words, and a hidden layer learnt by the system corresponding to the tags. While tagging the input data, we only observe the words—the tags (states) are hidden. States of the model are visible in training, not during the tagging task.

As discussed earlier, the HMM makes use of lexical and bi-gram probabilities estimated over a tagged training corpus in order to compute the most likely tag sequence for each sentence. One way to store the statistical information is to build a probability matrix. The probability matrix contains both the probability that an individual word belongs to a word class as well as the n-gram analysis, e.g., for a bi-gram model, the probability that a word of class X follows a word of class Y. This matrix is then used to drive the HMM tagger while tagging an unknown text.

We now return to the original problem. Given a sequence of words (sentence), the objective is to find the most probable tag sequence for the sentence.

Let W be the sequence of words.

$$W = w_1, w_2, \ldots, w_n$$

The task is to find the tag sequence

$$T = t_1, t_2, \ldots, t_n$$

which maximizes $P(T|W)$, i.e.,

$$T' = \text{argmax}_T\ P(T|W)$$

Applying Bayes Rule, $P(T|W)$ can be estimated using the expression:

$$P(T|W) = P(W|T) * P(T)/P(W)$$

As the probability of the word sequence, $P(W)$, remains the same for each tag sequence, we can drop it. The expression for the most likely tag sequence becomes:

$$T' = \text{argmax}_T\ P(W|T) * P(T)$$

Using the Markov assumption, the probability of a tag sequence can be estimated as the product of the probability of its constituent n-grams, i.e.,

$$P(T) = P(t_1) * P(t_2|t_1) * P(t_3|t_1t_2) \ldots * P(t_n|t_1 \ldots t_{n-1})$$

$P(W/T)$ is the probability of seeing a word sequence, given a tag sequence. For example, it is asking the probability of seeing 'The egg is rotten' given 'DT NNP VB JJ'. We make the following two assumptions:

- The words are independent of each other.
- The probability of a word is dependent only on its tag.

Using these assumptions, we obtain

$$P(W/T) = P(w_1/t_1) * P(w_2/t_2) \ldots P(w_i/t_i) * \ldots P(W_n/t_n)$$

i.e., $$P(W/T) = \prod_{i=1}^{n} P(w_i|t_i)$$

So, $$P(W/T) * P(T) = \prod_{i=1}^{n} P(w_i|t_i) \times P(t_1) \times P(t_2|t_1) * P(t_3|t_1t_2) \times \ldots \times P(t_n|t_1 \ldots t_{n-1})$$

Approximating the tag history using only the two previous tags, the transition probability, $P(T)$, becomes

$$P(T) = P(t_1) \times P(t_2|t_1) \times P(t_3|t_1t_2) \times \ldots \times P(t_n|t_{n-2}\ t_{n-1})$$

Hence, $P(T/W)$ can be estimated as

$$P(W/T) * P(T) = \prod_{i=1}^{n} P(w_i|t_i) \times P(t_1) \times P(t_2|t_1) * P(t_3|t_2t_1) \times \ldots \times P(t_n|t_{n-2}\ t_{n-1})$$

$$= \prod_{i=1}^{n} P(w_i|t_i) \times P(t_1) \times P(t_2/t_1) \times \prod_{i=3}^{n} P(t_i|t_{i-2}t_{i-1})$$

We estimate these probabilities from relative frequencies via Maximum Likelihood Estimation.

$$P(t_i|t_{i-2}\ t_{i-1}) = \frac{c(t_{i-2}, t_{i-1}, t_i)}{c(t_{i-2}, t_{i-1})}$$

$$P(w_i|t_i) = \frac{c(w_i, t_i)}{c(t_i)}$$

where $c(t_{i-2}, t_{i-1}, t_i)$ is the number of occurrences of t_i followed by $t_{i-2}\ t_{i-1}$.

Stochastic models have the advantage of being accurate and language independent. Most stochastic taggers have an accuracy of 96–97%. The accuracy seems to be quite high but it should be noted that this is measured as a percentage of words. An accuracy of 96% means that for a sentence containing 20 words, the error rate per sentence will be $1-0.96^{20} = 56\%$. This corresponds to approximately one word per sentence.

One of the drawbacks of stochastic taggers is that they require a manually tagged corpus for training. Kupiec (1992), Cutting *et al.* (1992), and others have demonstrated that the HMM tagger can be trained from unannotated text. This makes it possible to use the model for languages in which a manually tagged corpus is not available. However, a tagger trained on a hand-coded corpus performs better than one trained on an unannotated text. In order to achieve good performance a tagged corpus is required.

We now consider an example demonstrating how the probability of a particular part-of-speech sequence for a given sentence can be computed.

Example 3.3 Consider the sentence

The bird can fly.

and the tag sequence

DT NNP MD VB

Using bi-gram approximation, the probability

$$P\begin{pmatrix} \text{DT} & \text{NNP} & \text{MD} & \text{VB} \\ | & | & | & | \\ \text{The} & \text{bird} & \text{can} & \text{fly} \end{pmatrix}$$

can be computed as

$$= P(\text{DT}) \times P(\text{NNP}|\text{DT}) * P(\text{MD}|\text{NNP}) \times P(\text{VB}|\text{MD})$$
$$\times P(\text{the/DT}) \times P(\text{bird}|\text{NNP}) \times P(\text{can}|\text{MD}) \times P(\text{fly}|\text{VB})$$

3.7.3 Hybrid Taggers

Hybrid approaches to tagging combine the features of both the rule-based and stochastic approaches. They use rules to assign tags to words. Like the stochastic taggers, this is a machine learning technique and rules are automatically induced from the data. Transformation-based learning (TBL) of tags, also known as Brill tagging, is an example of hybrid approach. TBL is a machine learning method introduced by E. Brill (in 1995). Transformation-based error-driven learning has been applied to a number of natural language problems, including part-of-speech tagging, speech generation, and syntactic parsing (Brill 1993, 1994, Huang et al. 1994).

Figure 3.16 illustrates the TBL process. Like most HMM taggers, TBL is also a supervised learning technique. The steps involved in the TBL tagging algorithm are shown in Table 3.11. The input to Brill's TBL tagging algorithm is a tagged corpus and a lexicon (with most frequent information as indicated in the training corpus). The initial state annotator uses the lexicon to assign the most likely tag to each word as the start state. An ordered set of transformation rules are applied sequentially. The rule that results in the most improved tagging is selected. A manually tagged corpus is used as reference for truth. The process is iterated until some stopping criterion is reached, such as when no significant information is achieved over the previous iteration. At each iteration, the transformation that results in the highest score is selected. The output of the algorithm is a ranked list of learned transformation that transform the initial tagging close to the correct tagging. New text can then be annotated by first assigning the most frequent tag and then applying the ranked list of learned transformations in order.

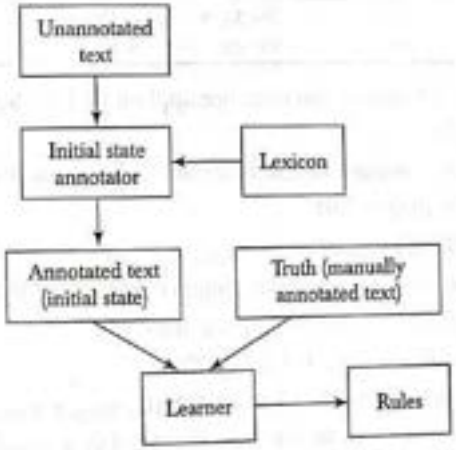


Figure 3.16 TBL learner

Table 3.11 TBL tagging algorithm

INPUT:	Tagged corpus and lexicon (with most frequent information)
Step 1:	Label every word with most likely tag (from dictionary)
Step 2:	Check every possible transformation and select one which most improves tagging
Step 3:	Re-tag corpus applying the rules
Repeat 2–3	Until some stopping criterion is reached
RESULT:	Ranked sequence of transformation rules

SUMMARY

This chapter has dealt with word level analysis. The topics covered include methods for characterizing word sequences, identifying morphological variants, detecting and correcting misspelled words, and identifying the correct part-of-speech for a word. The main points are as follows.

- Regular expressions can be used for specifying words. They can be encoded as a finite automaton.
- The goal of morphological parsing is to find out morphemes using which a given word is built. Morphemes are the smallest meaning-bearing units in a language.
- Morphological analysis and generation are essential to many NLP applications ranging from spelling error corrections to machine translations.
- The simplest morphological systems are stemmers. They do not use a lexicon. Instead, they use re-write rules. However, stemmers are not perfect.
- A two-level morphological model is more efficient. Both of its steps can be implemented using a finite state transducer.
- Word errors belong to one of two distinct categories, namely, *non-word errors* and *real word errors*. The latter category requires the context of the word to detect and correct errors.
- Words are classified into categories called part-of-speech or word classes. Word classes can be open or closed.
- Part-of-speech tagging is the process of assigning a part-of-speech like noun, verb, pronoun, preposition, adverb, adjective, etc., to each word in a sentence.
- Part-of-speech tagging methods fall under the following three categories:
  1. Rule-based (linguistic)
  2. Stochastic (data-driven)
  3. Hybrid

  Rule-based taggers use hand-coded rules to assign tags to words. *Stochastic taggers* require a pre-tagged corpus for training. *Hybrid taggers* combine features of both these approaches. Like rule-based systems, they use rules to specify tags. Like stochastic methods, they use machine learning to automatically induce rules from a tagged training corpus.
- Unknown words can be assigned the most frequent tags. We can also use morphological information to guess the correct tag.

REFERENCES

Brill, E., 1993, 'Transformation-based error-driven parsing,' *Proceedings of the Third International Workshop on Parsing Technologies*, Tilburg, The Netherlands.

Brill, E., 1994, 'Some advances in rule-based part of speech tagging,' *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, Seattle.

Brill, E., 1995, 'Error-driven learning and natural language processing: a case study in part-of-speech tagging,' *Computational Linguistics*.

Cutting, D., J. Kupiec, J. Pederson, and P. Sibun, 1992, 'A practical part-of-speech tagger,' *Proceedings of the Third Conference on Applied Natural Language Processing, ACL*.

Damerau, F. J., 1964, 'A technique for computer detection and correction of spelling errors,' *Communications of the ACM*, 7(3) pp. 171–76.

Francis, W. Nelson and Henry Kucera, 1982, *Frequency Analysis of English Usage: Lexicon and Grammar*, Houghton Mifflin, Boston.

Garside, R., G. Leech, and G. Sampson, 1987, *The Computational Analysis of English: A Corpus-Based Approach*, Longman, London.

Green, B. and G. Rubin, 1971, 'Automated grammatical tagging of English,' Department of Linguistics, Brown University.

Hardie, A., 2003, 'Developing a tag-set for automated part-of-speech tagging in Urdu,' *Proceedings of the Corpus Linguistics 2003 Conference*, D. Archer, P. Rayson, A. Wilson, and T. McEnery, (Eds.), UCREL Technical Papers, 16, Department of Linguistics, Lancaster University.

Hopcroft, J.E. and J.D. Ullman, 1979, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, Massachusetts.

Huang et al., 1994, 'Generation of pronunciation from orthographies using transformation-based error-driven learning,' *Proceedings of Int. Conference on Speech and Language Processing (ICSLP)*, Yokohama, Japan.

Kleene, S.C., 1956, 'Representation of events in nerve nets and finite automata,' *Automata Studies*, C. Shannon and J. McCarthy (Eds.), Princeton University Press, Princeton, NJ, pp. 3–41.

Koskenniemi, K., 1983, 'Two-level morphology: A general computational model of word-form recognition and production, Technical Report *Publication No. 11*, Department of General Linguistics,' University of Helsinki.

Krovetz, R., 1993, 'Viewing morphology as an inference process,' In *SIGIR-93*, pp. 191–202.

CHAPTER 4

SYNTACTIC ANALYSIS

CHAPTER OVERVIEW

This chapter introduces a number of important notions about syntax. Syntactic parsing deals with the syntactic structure of a sentence. In many languages, words are brought together to form larger groups termed constituents or phrases, which can be modelled using context-free grammar. Context-free grammar is a set of rules or productions that tell which elements can occur in a phrase and in what order. This chapter discusses phrase structural rules for various phrases and introduces feature structures to efficiently capture the properties of grammatical categories. The parsing strategies covered here include top-down and bottom-up parsing, dynamic programming parsing algorithms (namely, Earley and CYK), and probabilistic CYK parsing. Finally, a framework based on Paninian grammar is discussed for Hindi.

4.1 INTRODUCTION

In Chapter 3, we talked about word level analysis. We now move on to higher level constituents like phrases and sentences. The word 'syntax' refers to the grammatical arrangement of words in a sentence and their relationship with each other. The objective of syntactic analysis is to find the syntactic structure of the sentence. This structure is usually depicted as a tree, as shown in Figure 4.1. Nodes in the tree represent the phrases and leaves correspond to the words. The root of the tree is the whole sentence. Identifying the syntactic structure is useful in determining the meaning of the sentence. The identification is done using a process known as parsing. Syntactic parsing can be considered as the process of assigning 'phrase markers' to a sentence (Charniak 1997). One must, therefore, have a clear understanding of what phrases are. We will describe various phrases that could appear in a language so that we can specify grammar rules for them.

Researchers have proposed a number of parsing methods for natural language sentences. It is not possible to discuss all of them in a chapter, so we focus on a few widely-known ones. Most text books use only English for their discussions. We differ by including Indian languages as well. In addition to an introduction to grammar formalism, this chapter also provides an introduction to Hindi grammar.

Two important ideas in natural language are those of constituency and word order. Constituency is about how words are grouped together and how we know that they are really grouping together. Word order is about how, within a constituent, words are ordered with respect to one another, and also how constituents are ordered with respect to one another.

A widely used mathematical system for modelling constituent structure in natural language is context-free grammar (CFG) also known as phrase structure grammar. We begin our discussion with an overview of CFG in Section 4.2. In Section 4.3, we discuss various types of phrases and phrase structure rules. We then introduce feature structures in Section 4.4, to capture certain properties of grammatical categories which cannot be handled efficiently using CFG. We discuss probabilistic grammar and CYK parser based on this in Section 4.5. The CFG is basically a positional grammar and is not suitable for Indian languages, which are free word order languages. We provide a brief overview of Paninian grammar (PG), which is suitable for modelling free word order languages, in Section 4.6. We also discuss a parsing framework proposed by Bharti and Sangal (1990) for Indian languages based on this grammar. Finally, we provide a brief summary of the chapter.

4.2 CONTEXT-FREE GRAMMAR

Context-free grammar (CFG) was first defined for natural language by Chomsky (1957) and used for the Algol programming language by Backus (1959) and Naur (1960). A CFG (also called phrase-structure grammar) consists of four components:

1. A set of non-terminal symbols, N
2. A set of terminal symbols, T
3. A designated start symbol, S, that is one of the symbols from N.
4. A set of productions, P, of the form:

$$A \rightarrow \alpha$$

where $A \in N$ and α is a string consisting of terminal and non-terminal symbols. The rule $A \rightarrow \alpha$ says that constituent A can be rewritten as α. This is also called the phrase structure rule. It specifies which elements

(or constituents) can occur in a phrase and in what order. For example, the rule S → NP VP states that S consists of NP followed by VP, i.e., a sentence consists of a noun phrase followed by a verb phrase.

A language is usually defined through the concept of derivation. The basic operation is that of rewriting a symbol appearing on the left hand side of production by its right hand side. A CFG can be used to generate a sentence or to assign a structure to a given sentence. When used as a generator, the arrows in the production rule may be read as 'rewrite the symbol on the left with symbols on the right'. Consider the toy grammar shown in Figure 4.1. The symbol S can be rewritten as NP VP using Rule 1, then using rules R2 and R4, NP and VP are rewritten as N and V NP respectively. NP is then rewritten as Det N (R3). Finally, using rules R6 and R7, we get the sentence:

Hena reads a book. (4.1)

We say that the sentence (4.1) can be derived from S. The representation of this derivation is shown in Figure 4.1. Sometimes, a more compact bracketed notation is used to represent a parse tree. The parse tree in Figure 4.1 can be represented using this notation as follows:

$[_S [_{NP} [_N \text{Hena}]] [_{VP} [_V \text{reads}] [_{NP} [_{Det} \text{a}] [_N \text{book}]]]]$

The set of all the strings containing terminal symbols which can be derived from the start symbol of the grammar, defines the language generated by that grammar. The parse tree shown in Figure 4.1 essentially represents a mapping of a string to its parse tree. This mapping process is called parsing. We will come back to this issue in Section 4.4.

S → NP VP	R1	
NP → N	R2	
NP → Det N	R3	
VP → V NP	R4	
VP → V	R5	
N → Hena | She	R6	
V → reads|sings|sleeps	R7	

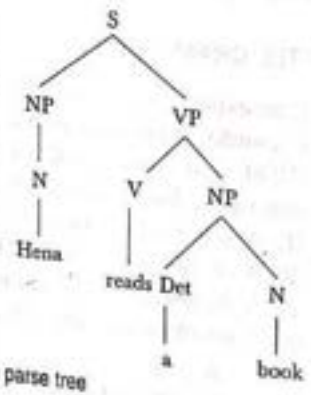


Figure 4.1 Toy CFG and sample parse tree

4.3 CONSTITUENCY

Words in a sentence are not tied together as a sequence of part-of-speech. Language puts constraints on word order. For example, certain words go together with each other more than with others, and seem to behave as a unit. The fundamental idea of syntax is that words group together to form constituents (often termed phrases), each of which acts as a single unit. They combine with other constituents to form larger constituents, and eventually, a sentence. *The bird, The rain, The Wimbledon court, The beautiful garden* are all noun phrases that can occur in the same syntactic context. For example, they can all function as the subject or the object of a verb. These constituents combine with others to form a sentence constituent. For example, the noun phrase, *The bird*, can combine with the verb phrase, *flies*, to form the sentence, *The bird flies*. Different types of phrases have different internal structures. In this section, we discuss some of the major phrase types and try to build phrase structure rules to identify them.

4.3.1 Phrase Level Constructions

As discussed earlier, a fundamental notion in natural language is that certain groups of words behave as constituents. These constituents are identified by their ability to occur in similar contexts. One of the simplest ways to decide whether a group of words is a phrase, is to see if it can be substituted with some other group of words without changing the meaning. If such a substitution is possible then the set of words forms a phrase. This is called the substitution test. Consider sentence (4.1). We can substitute a number of other phrases:

Hena reads a book.
Hena reads a storybook.
Those girls read a book.
She reads a comic book.

We can easily identify the constituents that can be replaced for each other in these sentences. These are *Hena, she*, and *Those girls* and *a book, a storybook*, and *a comic book*. These are the words that form a phrase. In linguistics, such constituents represent a paradigmatic relationship. Elements that can substitute each other in certain syntactic positions are said to be members of one paradigm.

Phrase types are named after their head, which is the lexical category that determines the properties of the phrase. Thus, if the head is a noun, the phrase is called a noun phrase, if the head is a verb, the phrase is

called a verb phrase; and so on for other lexical categories such as adjective and preposition. Figure 4.2 shows a sentence with a noun phrase, verb phrase, and preposition phrase.

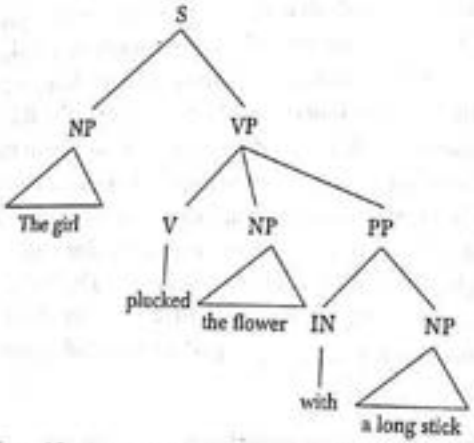


Figure 4.2 A sentence with NP, VP, and PP

Noun Phrase

A noun phrase is a phrase whose head is a noun or a pronoun, optionally accompanied by a set of modifiers. It can function as subject, object, or complement. The modifiers of a noun phrase can be determiners or adjective phrases. The obligatory constituent of a noun phrase is the noun head—all other constituents are optional. These structures can be represented using the phrase structure rule. As discussed earlier, phrase structure rules are of the form A → BC, which states that constituent A can be rewritten as two constituents B and C. These rules specify which elements can occur in a phrase and in what order. Using this notation, we can represent the phrase structure rules for a noun phrase as follows.

NP → Pronoun
NP → Det Noun
NP → Noun
NP → Adj Noun
NP → Det Adj Noun

We can combine all these rules in a single phrase structure rule as follows:

NP → (Det) (Adj) Noun|Pronoun

The constituents in parentheses are optional. This rule states that a noun phrase consists of a noun, possibly preceded by a determiner and an adjective (in that order). This rule does not cover all possible NPs. A

noun phrase may include post-modifiers and more than one adjective. For example, it may include a prepositional phrase (PP). More than one adjective is handled by allowing an adjective phrase (AP) for the adjective in the rule. After incorporating PP and AP in the phrase structure rule, we get the following.

NP → (Det) (AP) Noun (PP)

The following are a few examples of noun phrases:

They (4.2a)
The foggy morning (4.2b)
Chilled water (4.2c)
A beautiful lake in Kashmir (4.2d)
Cold banana shake (4.2e)

Let us see how the phrases (4.2a–e) can be generated using phrase structure rules. The phrase (4.2a) consists only of a pronoun; (4.2b) consists of a determiner, an adjective (foggy) that stands for an entire adjective phrase, and a noun; (4.2c) comprises an adjective phrase and a noun; (4.2d) consists of a determiner (the), an adjective phrase (beautiful), a noun (lake), and a prepositional phrase (in Kashmir); and (4.2e) consists of an adjective followed by a sequence of nouns. A noun sequence is termed as nominal. None of the phrase structure rules discussed so far are able to handle nominals. So, we modify our rules to cover this situation.

NP → (Det) (AP) Nom (PP)
Nom → Noun | Noun Nom

A noun phrase can act as a subject, an object, or a predicate. The following sentences demonstrate each of these uses.

The foggy damped weather disturbed the match. (4.3a)
I would like a *nice cold banana shake*. (4.3b)
Kula botanical garden is a *beautiful location*. (4.3c)

In (4.3a), the noun phrase acts as a subject. In (4.3b), it acts as an object, and in (4.3c), it is a predicate.

Verb Phrase

Analogous to the noun phrase is the verb phrase, which is headed by a verb. There is a fairly wide range of phrases that can modify a verb. This makes verb phrases a bit more complex. The verb phrase organizes various elements of the sentence that depend syntactically on the verb.

The following are some examples of verb phrases:

Khushbu *slept*. (4.4a)
The boy *kicked the ball*. (4.4b)

Khushbu slept in the garden. (4.4c)
The boy gave the girl a book. (4.4d)
The boy gave the girl a book with blue cover. (4.4e)

As you can see from these examples a verb phrase can have a verb [VP → Verb in (4.4a)]; a verb followed by an NP [VP → Verb NP in (4.4b)]; a verb followed by a PP [VP → Verb PP in (4.4c)]; a verb followed by two NPs [VP → Verb NP NP in (4.4d)]; or a verb followed by two NPs and a PP [VP → Verb NP NP PP in (4.4e)]. In general, the number of NPs in a VP is limited to two, whereas it is possible to add more than two PPs.

VP → Verb (NP) (NP) (PP)*

Things are further complicated by the fact that objects may also be entire clauses as in the sentence, *I know that Taj is one of the seven wonders.* Hence, we must also allow for an alternative phrase statement rule, in which NP is replaced by S.

VP → Verb S

Prepositional Phrase

Prepositional phrases are headed by a preposition. They consist of a preposition, possibly followed by some other constituent, usually a noun phrase.

We played volleyball on the beach.

We can have a preposition phrase that consists of just a preposition.

John went outside.

The phrase structure rule that captures the above eventualities is as follows.

PP → Prep (NP)

Adjective Phrase

The head of an adjective phrase (AP) is an adjective. APs consist of an adjective, which may be preceded by an adverb and followed by a PP. Here are few examples.

Ashish is clever.
The train is very late.
My sister is fond of animals.

The phrase structure rule for adjective phrase is

AP → (Adv) Adj (PP)

Adverb Phrase

An adverb phrase consists of an adverb, possibly preceded by a degree adverb. Here is an example.

Time passes very quickly.

AdvP → (Intens) Adv

4.3.2 Sentence Level Constructions

Having discussed phrase structures, we now focus our attention on sentences. A sentence can have varying structure. The four commonly known structures are declarative structure, imperative structure, yes-no question structure, and wh-question structure.

Sentences with a declarative structure have a subject followed by a predicate. The subject of a declarative sentence is a noun phrase and the predicate is a verb phrase, e.g., *I like horse riding.* The phrase structure rule for declarative sentences is

S → NP VP

Sentences with an imperative structure usually begin with a verb phrase and lack subject. The subject of these types of sentence is implicit and is understood to be 'you'. These types of sentences are used for commands and suggestions, and hence are called imperative. The grammar rule for this kind of sentence structure is

S → VP

Examples of this kind of sentences are as follows:

Look at the door.
Give me the book.
Stop talking.
Show me the latest design.

Sentences with the yes-no question structure ask questions which can be answered using yes or no. These sentences begin with an auxiliary verb, followed by a subject NP, followed by a VP. Here are some examples:

Do you have a red pen?
Is there a vacant quarter?
Is the game over?
Can you show me your album?

We expand our grammar by adding another rule for the expansion of S, as follows:

S → Aux NP VP

Sentences with wh-question structure are more complex. These sentences begin with a wh-words—who, which, where, what, why, and how. A wh-question may have a wh-phrase as a subject or may include another subject. Consider the following wh-question:

Which team won the match?

This sentence is similar to a declarative sentence except that it contains a wh-word. A simple rule to handle this type of sentence structure is

S → Wh-NP VP

Another type of wh-question structure is one that involves more than one NP. In this type of questions, the auxiliary verb comes before the subject NP, just as in yes-no question structures.

Which cameras can you show me in your shop?

The rule for this type of wh-questions is

S → Wh-NP Aux NP VP

A simplified view of the grammar rules discussed so far is summarized in Table 4.1.

Table 4.1 Summary of grammar rules

S	→	NP VP
S	→	VP
S	→	Aux NP VP
S	→	Wh-NP VP
S	→	Wh-NP Aux NP VP
NP	→	(Det) (AP) Nom (PP)
VP	→	Verb (NP) (NP) (PP)*
VP	→	Verb S
AP	→	(Adv) Adj (PP)
PP	→	Prep (NP)
Nom	→	

Coordination

The grammar rules in Table 4.1 are not exhaustive. There are other sentence-level structures that cannot be modelled by the rules discussed here. Coordination is one such structure. It refers to conjoining phrases with conjunctions like 'and', 'or', and 'but'. For example, a coordinate noun phrase can consist of two other noun phrases separated by a conjunction 'and', as in

I ate [NP [NP an apple] and [NP a banana]].

Similarly, verb phrases and prepositional phrases can be conjoined as follows:

It is [VP [VP dazzling] and [VP raining]].

Not only that, even a sentence can be conjoined.

[S [S I am reading the book] and [S I am also watching the movie]]

We need to devise rules to handle these constructions. Conjunction rules for NP, VP, and S can be built as follows:

NP → NP and NP
VP → VP and VP
S → S and S

Agreement

Most verbs use two different forms in present tense—one for third person, singular subjects, and the other for all other kinds of subjects. The third person singular (3sg) form ends with a -s whereas the non-3sg does not. Whenever there is a verb that has some noun acting as a subject, this agreement has to be confirmed. Here are a few examples that demonstrate how the subject NP affects the form of the verb.

Does [$_{NP}$ Priya] sing? (4.5a)
Do [$_{NP}$ they] eat? (4.5b)

In the first sentence, the subject NP is singular. Hence, the -es form of 'do', i.e. 'does' is used. The second sentence has a plural NP subject. Hence, the form 'do' is being used. Sentences in which subject and verb do not agree are ungrammatical. The following sentences are ungrammatical:

[Does] they eat? (4.5 c)
[Do] she sings? (4.5 d)

Sentences (4.5c) and (4.5d) point out that a grammatical phenomenon can lead to ungrammatical sentences—a problem known as over-generation. Rules that handle the yes-no questions of Example 4.5 are as follows:

S → Aux NP VP

To take care of the subject-verb agreement, we replace this rule with a pair of rules as follows:

S → 3sgAux 3sgNP VP
S → Non3sgAux Non3sgNP VP

These rules ensure appropriate subject-verb agreement. We could add rules for the lexicon like these:

3sg Aux → does | has | can
Non3sg Aux → do | have | can

Similarly, rules for 3sgNP and Non3sgNP need to be added. So we replace each of the phrase structure rules for noun phrase by a pair of rules as follows:

$$
\begin{aligned}
3sgNP &\rightarrow (Det)\ (AP)\ SgNom\ (PP)\\
Non3sgNP &\rightarrow (Det)\ (AP)\ PlNom\ (PP)\\
SgNom &\rightarrow SgNoun \mid SgNoun\ SgNom\\
PlNom &\rightarrow PlNoun \mid PlNoun\ PlNom\\
SgNoun &\rightarrow Priya \mid lake \mid banana \mid sister \mid \ldots\\
PlNoun &\rightarrow Children \mid \ldots
\end{aligned}
$$

We also have to add rules for the first and second person pronouns. This method of dealing with number agreement doubles the size of the grammar. Each rule that makes use of a noun or verb phrase results in the introduction of a pair of rules—one to handle singular form and another to handle plural form. We also need to introduce new versions of NP and noun rules for various cases, e.g., nominative (I, she, they, he) and accusative (me, her, him, them) cases of pronoun. Languages like Hindi and Urdu, which have not only noun-verb agreements but also gender agreements, further aggravate the problem by adding another multiplier. Clearly, CFG cannot handle this problem efficiently.

We solve the problem of over-generation by introducing new grammatical categories corresponding to each such constraint. This results in an explosion in the number of grammar rules and loss of generality. An alternative solution is to associate each non-terminal of the grammar with feature structures. Feature structures are able to capture grammatical properties without increasing the size of the grammar. We can think of grammatical categories (and the grammar rules that make use of them) as objects having properties associated with them. The information in these properties can be thought of as constraints imposed by the grammatical categories. Models based on this idea are called constraint-based formalisms. We now introduce feature structures and discuss how they are used to represent the constraints imposed by grammatical categories without the loss of generality.

Feature Structures

Feature structures are sets of feature-value pairs. They can be used to efficiently capture the properties of grammatical categories. Features are simply symbols representing properties that we wish to capture. For example, the number property of a noun phrase can be represented by NUMBER feature. The value that a NUMBER feature can take is SG (for singular) and PL (for plural). Values can be either atomic symbols or feature structures. Feature structures are represented by a matrix-like diagram called attribute value matrix (AVM).

$$
\begin{bmatrix} FEATURE_1 & VALUE_1 \\ FEATURE_2 & VALUE_2 \\ \ldots & \ldots \\ FEATURE_n & VALUE_n \end{bmatrix}
$$

Figure 4.3 An attribute value matrix (AVM)

An AVM consisting of a single NUMBER feature with the value SG is represented as follows:

$$[NUMBER\ \ SG]$$

The value of a feature can be left unspecified and represented by an empty pair of square brackets, as in the following example:

$$[NUMBER\ \ [\,]]$$

The feature structure can be used to encode the grammatical category of a constituent and the features associated with it. For example, the following structure represents the third person singular noun phrase.

$$\begin{bmatrix} CAT & NP \\ NUMBER & SG \\ PERSON & 3 \end{bmatrix}$$

Similarly, a third person plural noun phrase can be represented as follows:

$$\begin{bmatrix} CAT & NP \\ NUMBER & PL \\ PERSON & 3 \end{bmatrix}$$

The values of CAT and PERSON features remain the same in both structures. This explains how feature structures aid in generalization while making the necessary distinction possible. As mentioned earlier, feature values are not limited to atomic symbols. A feature can have another feature structure as its value. Consider the case of combining the NUMBER and PERSON features into a single AGREEMENT feature. This makes sense because grammatical subjects must agree with their predicates in NUMBER as well as PERSON properties. Using this new feature, we represent the grammatical category third person plural noun phrase by the following structure:

$$\begin{bmatrix} CAT & NP \\ AGREEMRENT & \begin{bmatrix} NUMBER & PL \\ PERSON & 3 \end{bmatrix} \end{bmatrix}$$

In order for feature structures to be useful, we must be able to perform operations on them. The two most important operations we need to perform are merging the information content of the two structures that are similar and rejecting structures that are incompatible. The computational technique that is used to perform these operations is called unification. Unification is implemented as a binary operator ($\sqcup$) that takes two feature structures as arguments and returns a merged feature structure if they are compatible, otherwise reports a failure. Here is a simple application of the unification operator for performing an equality check.

$$[NUMBER \quad PL] \sqcup [NUMBER \quad PL] = [NUMBER \quad PL]$$

The unification succeeds as the two structures have the same value for the NUMBER feature. A feature with an unspecified value in one structure, can be successfully matched with any value in a corresponding feature in another structure. In such cases, the unification operation produces a structure with the value provided by the structure having non-null value. For example,

$$[NUMBER \quad PL] \sqcup [NUMBER \quad []] = [NUMBER \quad PL]$$

In this example, the two structures are considered compatible and merged into a structure with PL as its value for the NUMBER feature. The value PL of the first structure matches the value [] of the second structure and becomes the value of the NUMBER feature of the output structure.

However, the following application of unification results in failure as the NUMBER features of the first and second structures have incompatible values.

$$[NUMBER \quad PL] \sqcup [NUMBER \quad SG] \text{ Fails}$$

The CFG rules can have feature structures attached to them to realize constraints on the constituents of the sentence.

4.4 PARSING

A CFG defines the syntax of a language but does not specify how structures are assigned. The task that uses the rewrite rules of a grammar to either generate a particular sequence of words or reconstruct its derivation (or phrase structure tree) is termed parsing. A phrase structure tree constructed from a sentence is called a parse. The syntactic parser is thus responsible for recognizing a sentence and assigning a syntactic structure to it. It is possible for many different phrase structure trees to derive the same sequence of words. This means a sentence can have multiple parses. This phenomenon is called syntactic ambiguity.

Garden pathing is another phenomenon related to syntactic parsing. It refers to the process of constructing a parse by exploring the parse tree along different paths, one after the other till, eventually, the right one is found. The popular example given for garden pathing is the sentence

The horse ran past the barn fell. (4.6)

In the first attempt, most of us come up with a parse corresponding to the sentence *The horse ran past the barn*, leaving no possibility for the word *fell* to be incrementally added in the sentence. In order to complete the parse of the sentence, we have to backtrack.

Finding the right parse can be viewed as a search process. The search finds all trees whose root is the start symbol S and whose leaves cover exactly the word in the input. The search space in this conception corresponds to all possible parse trees defined by the grammar. The following constraints guide the search process.

1. *Input*: The first constraint comes from the words in the input sentence. A valid parse is one that covers all the words in a sentence. Hence, these words must constitute the leaves of the final parse tree.
2. *Grammar*: The second kind of constraint comes from the grammar. The root of the final parse tree must be the start symbol of the grammar.

These two constraints give rise to the two most widely used search strategies by parsers, namely, top-down or goal-directed search and bottom-up or data-directed search.

4.4.1 Top-down Parsing

As the name suggests, top-down parsing starts its search from the root node S and works downwards towards the leaves. The underlying assumption here is that the input can be derived from the designated start symbol, S, of the grammar. The next step is to find all sub-trees which can start with S. To generate the sub-trees of the second-level search, we expand the root node using all the grammar rules with S on their left hand side. Likewise, each non-terminal symbol in the resulting sub-trees is expanded next using the grammar rules having a matching non-terminal symbol on their left hand side. The right hand side of the grammar rules provide the nodes to be generated, which are then expanded recursively. As the expansion continues, the tree grows downward and eventually reaches a state where the bottom of the tree consist only of part-of-speech categories. At this point, all trees whose leaves do not match words in the input sentence are rejected, leaving only trees that represent successful

parses. A successful parse corresponds to a tree which matches exactly with the words in the input sentence.

Table 4.2 Sample grammar

S → NP VP	VP → Verb NP
S → VP	VP → Verb
NP → Det Nominal	PP → Preposition NP
NP → Noun	Det → this | that | a | the
NP → Det Noun PP	Verb → sleeps | sings | open | saw | paint
Nominal → Noun	Preposition → from | with | on | to
Nominal → Noun Nominal	Pronoun → she | he | they

Consider the grammar shown in Table 4.2 and the sentence

Paint the door. (4.7)

A top-down search begins with the start symbol of the grammar. Thus, the first level (ply) search tree consists of a single node labelled S. The grammar in Table 4.2 has two rules with S on their left hand side. These

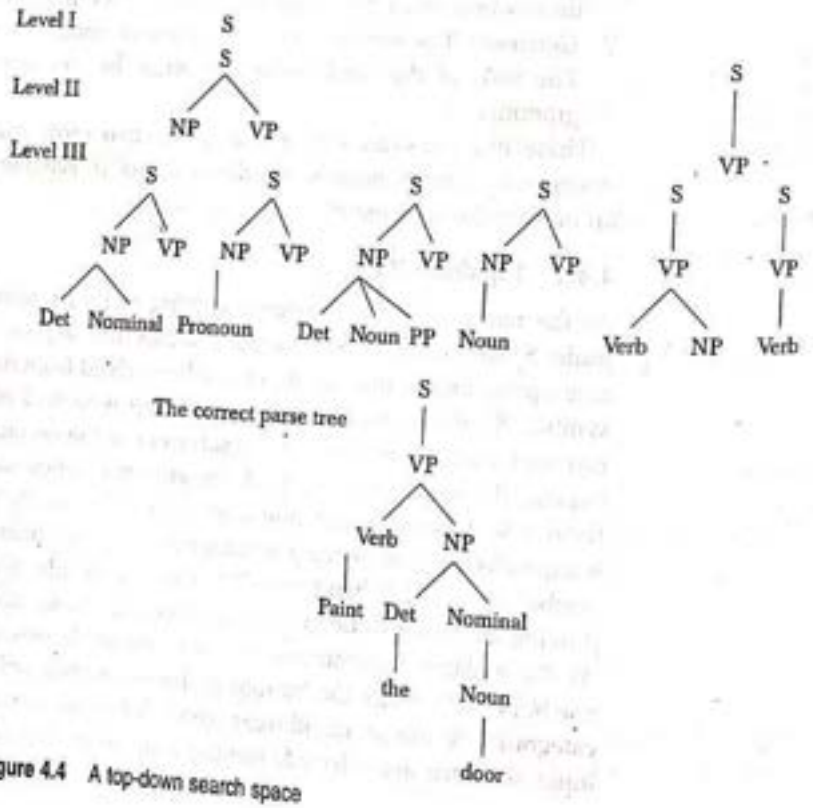


Figure 4.4 A top-down search space

rules are used to expand the tree, which gives us two partial trees at the second level search, as shown in Figure 4.4. The third level is generated by expanding the non-terminal at the bottom of the search tree in the previous ply. Due to space constraints, only the expansion corresponding to the left-most non-terminals has been shown in the figure. The subsequent steps in the parse are left, as an exercise, to the readers. The correct parse tree shown in Figure 4.4 is obtained by expanding the fifth parse tree of the third level.

4.4.2 Bottom-up Parsing

A bottom-up parser starts with the words in the input sentence and attempts to construct a parse tree in an upward direction towards the root. At each step, the parser looks for rules in the grammar where the right hand side matches some of the portions in the parse tree constructed so far, and reduces it using the left hand side of the production. The parse is considered successful if the parser reduces the tree to the start symbol of the grammar. Figure 4.5 shows some steps carried out by the bottom-up parser for sentence (4.7).

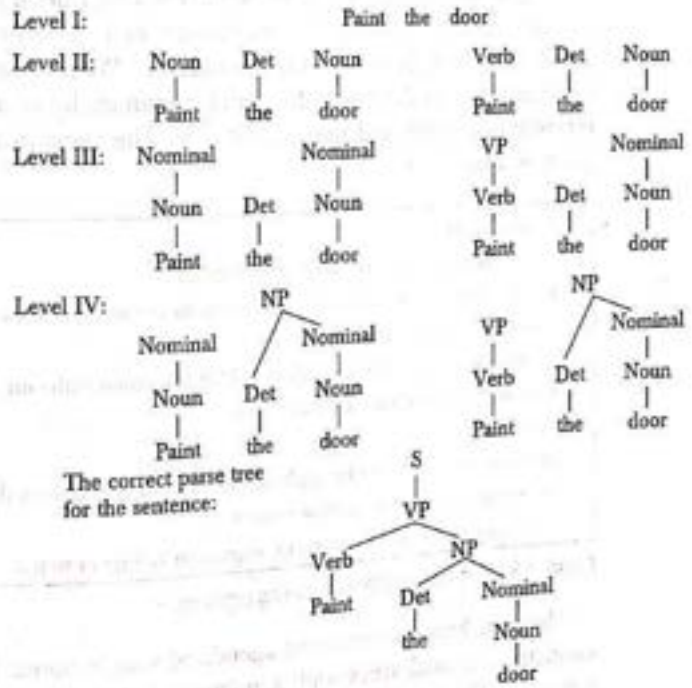


Figure 4.5 A bottom-up search space for sentence (4.7)

Each of these parsing strategies has its advantages and disadvantages. As the top-down search starts generating trees with the start symbol of the grammar, it never wastes time exploring a tree leading to a different root. However, it wastes considerable time exploring S trees that eventually result in words that are inconsistent with the input. This is because a top-down parser generates trees before seeing the input. On the other hand, a bottom-up parser never explores a tree that does not match the input. However, it wastes time generating trees that have no chance of leading to an S-rooted tree. The left branch of the search space in Figure 4.5 that explores a sub-tree assuming *paint* as a noun, is an example of wasted effort. We now present a basic search strategy that uses the top-down method to generate trees and augments it with bottom-up constraints to filter bad parses.

4.4.3 A Basic Top-down Parser

The approach presented here is essentially a depth first, left to right search. The depth first approach expands the search space incrementally by one state at a time. At each step, the left-most unexpanded leaf nodes of the tree are expanded first using the relevant rule of the grammar. The left-most node is selected for expansion as it determines the order in which input words needs to be considered. When a state arrives that is inconsistent with the input, the search continues by returning to the most recently generated and unexplored tree. The steps of the algorithm are given in Figure 4.6.

1. Initialize agenda
2. Pick a state, let it be curr_state, from agenda
3. If (curr_state) represents a successful parse then return parse tree
   else if curr_stat is a POS then
   if category of curr_state is a subset of POS associated with curr_word
   then apply lexical rules to current state
   else reject
   else generate new states by applying grammar rules and push them into agenda
4. If (agenda is empty) then return failure
   else select a node from agenda for expansion and go to step 3.

Figure 4.6 Top-down, depth-first parsing algorithm

The algorithm maintains an agenda of search states. Each search state consists of partial trees and a pointer to the next input word in the sentence. The algorithm starts with the state at the front of the agenda

and generates a set of new states by applying grammar rule to the left-most unexpanded node of the tree associated with it. The newly generated states are put on the front of the agenda in the order defined by the textual order of the grammar rules used to create them. The process continues until either a successful parse tree is discovered or the agenda is empty, indicating a failure.

Figure 4.7 shows the trace of the algorithm on the sentence, *Open the door*. The algorithm starts with the node S and input word *Open*. It first expands S using the grammar rule S → NP VP. It then expands the left-most unexpanded non-terminal NP using the rule NP → Det Nominal. But the word *Open* cannot be derived from Det. Hence, the parser eliminates the rule and tries the second alternative, i.e., NP → noun, which again leads to a failure. The next search space on the agenda corresponds to the S → VP rule. The expansion of VP using the rule VP → Verb NP, successfully matches the first input words. The algorithm proceeds in a depth-first, left-to-right manner, to match the rest of the input words.

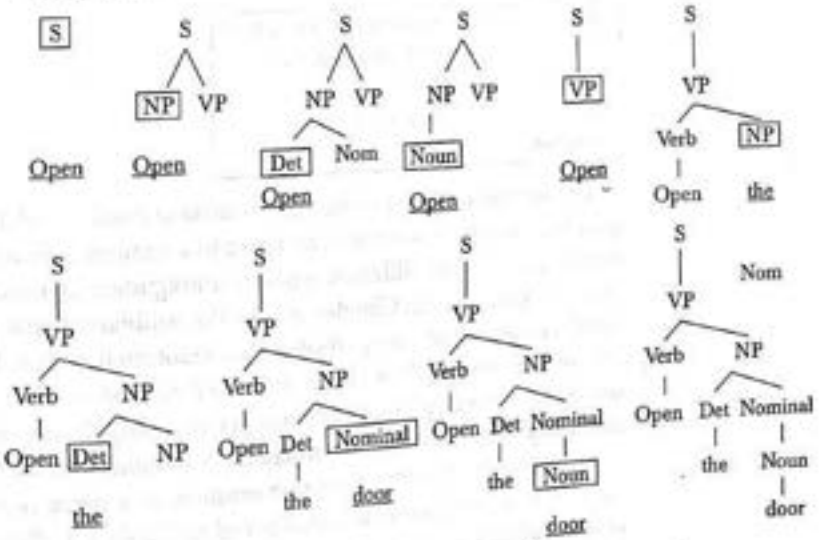


Figure 4.7 A derivation using top-down, depth-first algorithm

In any successful parse, the current input word must match the first word in the derivation of the node that is being expanded. This information can be utilized in eliminating spurious parses. A grammar rule that cannot lead to the input word as the first word along the left side of a derivation, should not be considered for expansion. The first word along the left side

of the derivation is called the left corner of the tree. Using the left corner notion, we see that in our example, only the rule S → VP is applicable, as the word *Open* cannot be a left corner of the NP. In order to utilize this filter, we create a table containing a list of all the valid left corner categories for each non-terminal of the grammar. While selecting a rule for expansion, the table is consulted to see if the non-terminal associated with the rule has a part-of-speech associated with the current input. If not, then the rule is not considered. The left corner table for grammar is shown in Table 4.3.

The top-down, depth-first, left-to-right search algorithm suffers from certain disadvantages. The first is that of left recursion, which causes the search to get stuck in an infinite loop. This problem arises if the grammar is left recursive that, is, it contains a non-terminal A, which derives, in one or more steps, a string beginning with the same non-terminal, i.e., $A^* \Rightarrow A\beta$ for some β.

Table 4.3 Left corner for each grammar category

Category	Left Corners
S	Det, Pronoun, Noun, Verb
NP	Noun, Pronoun, Det
VP	Verb
PP	Preposition
Nominal	Noun

The second problem is that of *structural ambiguity*, which occurs when a grammar assigns more than one parse to a sentence. Structural ambiguity is just one of many different types of ambiguities that arise in NLP. The one we discussed in Chapter 3 was the ambiguity that occurs when a word has more than one part-of-speech associated with it. In the following chapter, we review these ambiguities and discuss various disambiguation strategies. Structural ambiguity occurs in many forms, e.g., attachment ambiguity and coordination ambiguity. A sentence has *attachment ambiguity* if a constituent fits more than one position in a parse tree. For example, according to the grammar summarized in Table 4.1, there are two ways of generating the prepositional phrase, *with a long stick*, in the sentence, *The girl plucked the flower with a long stick*. It can be generated from the verb phrase, as in the parse tree shown in Figure 4.2, or from the noun phrase, as in the parse tree shown in Figure 4.8. The first parse leads to the interpretation that the stick is used to pluck the flower, whereas the second parse gives the interpretation that the flower being plucked has a long stick.

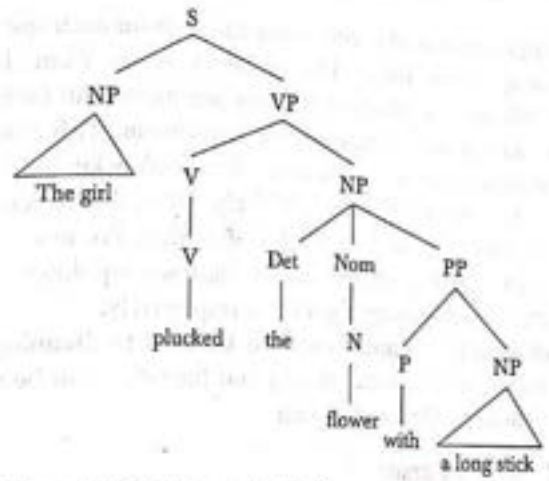


Figure 4.8 PP-attachment ambiguity

Coordination ambiguity occurs when it is not clear which phrases are being combined with a conjunction like *and*. For example, the phrase *beautiful hair and eyes* may have the structure [*beautiful hair*] and [*eyes*] or [*beautiful hair*] and [*beautiful eyes*]. Identifying the correct parse from a number of possible parses is known as disambiguation. A parser may utilize statistical and semantic knowledge to disambiguate the parse tree, or it may return all possible parses and leave the disambiguation for subsequent processing. The basic top-down parser we discussed, returns the first successful parse without exploring other possibilities. It needs to be modified to return all possible parses.

A sentence may have *local ambiguity* resulting in inefficient parsing. Local ambiguity occurs when certain parts of a sentence are ambiguous. For example, the sentence *Paint the door* is unambiguous, but during parsing it is not known whether the first word *Paint* is a verb or a noun. Hence, the parser makes a few incorrect expansions before discovering that Paint is a verb. Thus, it must use backtracking, or parallelism, to consider both parses.

Yet another problem associated with our basic top-down strategy is that of repeated parsing. The parser often builds valid trees for portions of the input that it discards during backtracking. These have to be rebuilt during subsequent steps in the parse. If we could avoid this extra effort, we would have more efficient parsers.

Dynamic programming algorithms can solve these problems. These algorithms construct a table containing solutions to sub-problems, which, if solved, will solve the whole problem. In parsing, a dynamic programming

algorithm builds a table containing sub-trees for each and every constituent appearing in the input. The efficiency comes from the fact that once these sub-trees are discovered, they are stored and later consulted by all parses attempting to expand that constituent. This solves the reparsing and ambiguity problem. There are three widely known dynamic parsers—the Cocke-Younger-Kasami (CYK) algorithm, the Graham-Harrison-Ruzzo (GHR) algorithm, and the Earley algorithm. We now present the Earley and CYK parsing algorithms to illustrate top-down and a bottom-up dynamic programming algorithms respectively.

Probabilistic grammar can also be used to disambiguate parse trees. We discuss probabilistic parsing and how they can be used to identify a likely parse, in the next section.

4.4.4 Earley Parser

The Earley parser implements an efficient parallel top-down search using dynamic programming. It builds a table of sub-trees for each of the constituents in the input. This way, the algorithm eliminates the repetitive parse of a constituent which arises from backtracking, and successfully reduces the exponential-time problem to polynomial time. The Earley parser can handle recursive rules such as A → AC without getting into an infinite loop.

The most important component of this algorithm is the Earley chart that has n+1 entries, where n is the number of words in the input. The chart contains a set of states for each word position in the sentence. The algorithm makes a left to right scan of input to fill the elements in this chart. It builds a set of states, one for each position in the input string (starting from 0), that describe the condition of the recognition process at that point in the scan. The states in each entry provide the following information.

1. A sub-tree corresponding to a grammar rule.
2. Information about the progress made in completing the sub-tree.
3. Position of the sub-tree with respect to input.

A state is represented as a dotted rule and a pair of numbers representing starting position and the position of dot. This representation takes the form

$$A \rightarrow X_1 \ldots \bullet C \ldots X_m, [i, j]$$

where the dot (•) represents the position in the rule's right hand side, and the two numbers (i and j) represent where the state begins and where the dot lies. A dot at the right end of the rule represents a successful parse of the associated non-terminal.


```
Earley Parsing
Input: Sentence and the Grammar
Output: Chart
    chart[0] ← S' → S, [0,0]
    n ← length (sentence) // number of words in the sentence
    for i = 0 to n do
        for each state in chart[i] do
        if (incomplete (state) and next category is not a part of speech) then
        predictor (state)
        else if (incomplete (state) and next category is a part of speech)
                scanner (state)
            else
                completer (state)
            end-if
        end-if
    end for
end for
return
Procedure predictor (A → X1 ... •B...Xm, [i, j])
for each rule (B → α) in G do
insert  the state B → • α, [j, j] to chart [j]
End
Procedure scanner (A → X1 ... •B...Xm, [i, j])
If B is one of the part of speech associated with word[j] then
Insert the state B → word [j] •, [j, j + 1] to chart [j + 1]
End
Procedure Completer (A → X1 ... •, [j, k])
for each B → X1...•A ...,[i, j] in chart[j] do
insert the state B → X1...A• ...[i, k] to chart[k]
End
```

Figure 4.9 The Earley Parsing Algorithm

The algorithm uses three operations to process states in the chart. These are:

- Predictor
- Scanner
- Completer

The algorithm sequentially constructs the sets for each of the n+1 chart entries. Chart [0] is initialized with a dummy state S' → •S, [0,0]. At each step one of the three operations are applicable depending on the state.

Application of these operators result in addition of new states to either the current or the next set of states. The presence of a state $S \rightarrow \alpha$, [0,N] indicates a successful parse. The algorithm is shown in Figure 4.9.

We now explain the function of the three operators.

Predictor

As the name suggests, the predictor generates new states representing potential expansion of the non-terminal in the left-most derivation. A predictor is applied to every state that has a non-terminal to the right of the dot, when the category of that non-terminal is different from the part-of-speech. The application of this operator results in the creation of as many new states as there are grammar rules for the non-terminal. These new states are placed into the same chart entry as the generating state. Their start and end positions are at the point where the generating state ends. If

$$A \rightarrow X_1 \ldots \bullet B \ldots X_m, [i, j]$$

Then for every rule of the form $B \rightarrow \alpha$, the operation adds to chart [j], the state

$$B \rightarrow \bullet\ \alpha, [j, j]$$

For example, when the generating state is S → • NP VP, [0,0], the predictor adds the following states to chart [0]:

NP → • Det Nominal, [0,0]
NP → • Noun, [0,0]
NP → • Pronoun, [0,0]
NP → • Det Noun PP, [0,0]

Scanner

A scanner is used when a state has a part-of-speech category to the right of the dot. The scanner examines the input to see if the part-of-speech appearing to the right of the dot matches one of the part-of-speech associated with the current input. If yes, then it creates a new state using the rule that allows generation of the input word with this part-of-speech. It advances the pointer over the predicted input category and adds it to the next chart entry. If the state is A → ...• a ..., [i, j] and 'a' is one of the part-of-speech associated with w_j then it adds a → ... w_j •,[i, j] to chart [j+1].

Returning to our example, when the state NP → • Det Nominal, [0,0] is processed, the parser finds a part-of-speech category next to the dot. It checks if the category of the current word (curr_word) matches with the expectation in the current state. If yes, then it adds the new state Det → curr_word •, [0,1] to the next chart entry.

Completer

The completer is used when the dot reaches the right end of the rule. The presence of such a state signifies successful completion of the parse of some grammatical category. The completer identifies all previously generated states that expect this grammatical category at this position in the input and creates new states by advancing the dots over the expected category. All these newly generated states are inserted in the current chart entry. More formally, if A → ...•, [j,k], then the completer adds B → ...A• ...[i,k] to chart [k] for all states B → ...•A ...,[i, j] in chart [j]. An item is added to a set only if it is not already in the set.

Example 4.1 Let us trace the algorithm using sentence (4.7). The sequence of states created by the parser is shown in Figure 4.10.

Chart [0] Dummy	S0	S' → • S start	[0,0] state
	S1	S → • NP VP	
	S2	S → • VP	
	S3	NP → • Det Nominal	
	S4	NP → • Noun	
	S5	NP → • Pronoun	
	S6	NP → Det Noun PP	
	S7	VP → • Verb NP	
	S8	VP → • Verb	
Chart [1]	S9	Noun → paint •	[0,1]
	S10	Verb → paint •	[0,1]
	S11	NP → Noun •	[0,1]
	S12	VP → Verb• NP	[0,1]
	S13	VP → Verb •	[0,1]
	S14	S → NP • VP	[0,1]
	S15	NP → • Det Nominal	[1,1]
	S16	NP → • Noun	[1,1]
	S17	S → VP •	[0,1]
	S18	VP → • Verb NP	[1,1]
	S19	VP → • Verb	[1,1]
Chart [2]	S20	Det → the •	[1,2]
	S21	NP → Det • Nominal	[1,2]
	S22	Nominal → • Noun	[2,2]
	S23	Nominal → • Noun Nominal	[2,2]
Chart [3]	S24	Noun → door •	[2,3]
	S25	Nominal → Noun •	[2,3]
	S26	NP → Det Nominal•	[1,3]
	S27	S → NP • VP	[0,3]
	S28	VP → Verb NP •	[0,3]
	S29	VP → • Verb NP	[3,3]
	S30	VP → • Verb	[3,3]
	S31	S → VP •	[0,3]

Figure 4.10 Sequence of states created in parsing sentence (4.7) using Earley algorithm

The presence of the state S → VP •, [0,3] in the chart indicates successful completion of the parse of the sentence, but it does not return the exact parse. Thus, the algorithm in this form can be used only to recognize a sentence. However, it is possible to utilize the entries appearing in the table to construct the parse tree of a valid sentence. In order to do so the algorithm needs to be modified. This modification is carried out by the completer which creates new states by advancing earlier incomplete states whenever it finds a state having a dot at the end of the rule. We add a pointer to the previous states of the new state. A list of previous states is thus maintained. Extracting a parse tree from the chart involves following these pointers, beginning with the state marking the successful completion of the parse. Though the Earley algorithm fills the chart entry in polynomial time, extracting all parse trees still requires an exponential amount of time.

The Earley parser can be augmented with unification structures to eliminate ill-formed structures as they are introduced. This requires certain modification in the algorithm. The first change involves the addition of a feature structure derived from their unification constraints. The second change is the addition of a new field to the chart. This new field is a directed acyclic graph representing the feature structure corresponding to the state. The details of the modified Earley algorithm can be found in Jurafsky and Martin (2000).

4.4.5 The CYK Parser

Like the Earley algorithm, the CYK (Cocke-Younger-Kasami) is a dynamic programming parsing algorithm. However, it follows a bottom-up approach in parsing. It builds a parse tree incrementally. Each entry in the table is based on previous entries. The process is iterated until the entire sentence has been parsed. The CYK parsing algorithm assumes the grammar to be in Chomsky normal form (CNF). A CFG is in CNF if all the rules are of only two forms:

A → BC

A → w, where w is a word.

The algorithm first builds parse trees of length one by considering all rules which could produce words in the sentence being parsed. Then, it constructs the most probable parse for all the constituents of length two. The parse of shorter constituents constructed in earlier iterations can be used in constructing the parse of longer constituents.

Like the Earley algorithm, the basic CYK algorithm is also a chart-based algorithm. A non-terminal is stored in the $[i, j]$th entry of the chart if, and only if, $A \Rightarrow w_i \cdot w_{i+1} \cdots w_{i+j-1}$. The chart is triangular. A sentence is recognized if the start symbol S is in the entry $[1, n]$ of the chart. Beginning with the start symbol of the grammar, we are able to derive the entire sequence of words appearing in the sentence. The algorithm builds smaller constituents before attempting to construct larger ones. First, the terminal derivation rules of the grammar are used to generate the $[i,1]$th entries. These entries represent non-terminals that derive the individual words appearing in the sentence, $w_{i1} = w_i$ for $1 \le i \le n$, where n is the length (number of words) of the sentence. $A \Rightarrow w_{i1}$ if $A \rightarrow w_i$ is a rule in the grammar. It then continues with sub-string of length two, three, and so on. For every non-terminal A in the grammar, the algorithm determines if $A^* \Rightarrow w_{ij}$. As the grammar is in CNF, A could derive if there existed a rule of the form A → B C such that B derives the first k words of the w_{ij} (i.e. $B \rightarrow w_{ik}$) and C derives the remaining j-k words ($C \rightarrow w_{kj}$) as shown in Figure 4.11. More formally,

$A^* \Rightarrow w_{ij}$ if

1. A → B C is a rule in grammar
2. $B^* \Rightarrow w_{ik}$ and
3. $C^* \Rightarrow w_{kj}$

For a sub-string w_{ij} of length j starting at i, the algorithm considers all possible ways of breaking it into two parts w_{ik} and w_{kj}. Finally, since s = w_{1n}, we have to verify that $S^* \Rightarrow w_{1n}$ i.e., the start symbol of the grammar derives w_{1n}.

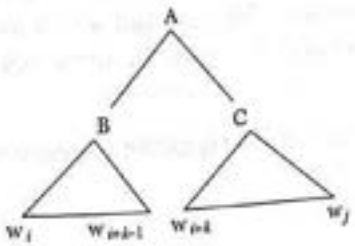


Figure 4.11 Breaking a string

The steps involved in the algorithm are shown in Figure 4.12. In order to use the information contained in the table to construct parse trees, we need to maintain back-pointers to the table entries we combine. This allows us to construct all possible parse trees by following back pointers.

```
Let w = w1 w2 w3 wi...wj...wn
  and wij = wi...wi+j-1
// Initialization step
for i := 1 to n do
   for all rules A→ wi do
      chart [i,1] = {A}
// Recursive step
for j = 2 to n do
   for i = 1 to n-j+1 do
   begin
      chart [i, j] = φ
      for k = 1 to j-1 do
      chart [i, j] := chart[i, j] ∪{A | A → BC is a production and
                        B ∈ chart[i, k] and C ∈ chart [i+k, j-k]}
   end
if S ∈ chart[1, n] then accept else reject
```

Figure 4.12 The CYK algorithm

To give a better understanding of the whole idea, we work out an example. Consider the following simplified grammar in CNF:

S → NP VP	Verb → wrote
VP → Verb NP	Noun → girl
NP → Det Noun	Noun → essay
Det → an | the	

The sentence to be parsed is: *The girl wrote an essay.*

Table 4.4 contains entries after a complete scan of the algorithm. The entry in the [1,n]th cell contains a start symbol which indicates that $S^* \Rightarrow w_{1n}$ i.e., the parse is successful. It is possible for a cell to have multiple entries.

Table 4.4 Sequence of states created in the chart by the CYK algorithm while parsing the sentence, *Sana wrote an essay*

	1	2	3	4	5
1	Det → The	NP → Det Noun			S → NP VP
2	Noun → Girl				
3	Verb → wrote		VP → Verb NP		
4	Det → an	NP → Det Noun			
5	Noun → essay				

4.5 PROBABILISTIC PARSING

Statistical parser, like statistical tagging (Chapter 3), requires a corpus of hand-parsed text. There are such corpora available, the most notably being the Penn tree-bank (Marcus et al. 1993). The Penn tree-bank is a large corpus of articles from the *Wall Street Journal* that have been tagged with Penn tree-bank tags and then parsed according to a simple set of phrase structure rules conforming to Chomsky's government and binding syntax. The parsed sentences are represented in the form of properly bracketed trees. A statistical parser works by assigning probabilities to possible parses of a sentence and returning the most likely parse as the final one. More formally, given a grammar G, sentence s, and a set of possible parse trees of s which we denote by $\tau(s)$, a probabilistic parser finds the most likely parse φ' of s as follows:

$$\begin{aligned}\varphi' &= \text{argmax}_{\varphi \in \tau(s)} P(\varphi \mid s)\\ &= \text{argmax}_{\varphi \in \tau(s)} P(\varphi, s)\\ &= \text{argmax}_{\varphi \in \tau(s)} P(\varphi)\end{aligned}$$

In order to construct a statistical parser, we have to first find all possible parses of a sentence, then assign probabilities to them, and finally return the most probable parse. We discuss an implementation of statistical parsing based upon probabilistic context-free grammars (PCFGs). But before proceeding ahead to this discussion, let us have a look at why we need probabilistic parsing at all. What advantages do these parsers offer?

The first benefit that a probabilistic parser offers is removal of ambiguity from parsing. We have seen earlier, that sentences can have multiple parse trees. The parsing algorithm discussed so far in this chapter has no means to decide which parse is the correct or most appropriate one. Probabilistic parsers assign probabilities to parses. These probabilities are then used to decide the most likely parse tree structure of an input sentence.

Another benefit this parser offers is related to efficiency. The search space of possible tree structures is usually very large. With no information on which sub-trees are more likely to be a part of the final parse tree, the search can be quite time consuming. Using probabilities to guide the process, the search becomes more efficient.

A probabilistic context-free grammar (PCFG) is a CFG in which every rule is assigned a probability (Charniak 1993). It extends the CFG by augmenting each rule $A \rightarrow \alpha$ in set of productions P, with a conditional probability p:

$$A \rightarrow \alpha\ [p]$$

where p gives the probability of expanding a constituent using the rule. $A \rightarrow \alpha$.

Let us now define PCFG. A PCFG is defined by the pair (G, f), where G is a CFG and f is a positive function defined over the set of rules such that, the sum of the probabilities associated with the rules expanding a particular non-terminal is 1 (Infante-Lopez and Maarten de Rijke 2006).

$$\sum_{\alpha} f(A \rightarrow \alpha) = 1$$

An example of PCFG is shown in Figure 4.13. We can verify that for each non-terminal, the sum of probabilities is 1.

f(S → NP VP) + f(S → VP) = 1

f(NP → Det Noun) + f(NP → Noun)+ f(NP → Pronoun) + f(NP → Det Noun PP) = 1

f(VP → Verb NP) + f(NP → Verb) + f(VP → VP PP) = 1.0

f(Det → this) + f(Det → that) + f(Det → a) + f(Det → the) = 1.0

f(Noun → paint) + f(Noun → door) + f(Noun → bird) + f(Noun → hole) = 1.0

S → NP VP	0.8	Noun → door	0.25
S → VP	0.2	Noun → bird	0.25
NP→ Det Noun	0.4	Noun → hole	0.25
NP → Noun	0.2	Verb → sleeps	0.2
NP → Pronoun	0.2	Verb → sings	0.2
NP → Det Noun PP	0.2	Verb → open	0.2
VP → Verb NP	0.5	Verb → saw	0.2
VP → Verb	0.3	Verb → paint	0.2
VP → VP PP	0.2	Preposition → from	0.3
PP → Preposition NP	1.0	Preposition → with	0.25
Det → this	0.2	Preposition → on	0.2
Det → that	0.2	Preposition → to	0.25
Det → a	0.25	Pronoun→ she	0.35
Det → the	0.35	Pronoun→ he	0.35
Noun → paint	0.25	Pronoun→ they	0.25

Figure 4.13 A probabilistic context-free grammar (PCFG)

Similarly, the condition is satisfied for the other non-terminals.

4.5.1 Estimating Rule Probabilities

The next question is how are probabilities assigned to rules? One way to estimate probabilities for a PCFG is to manually construct a corpus of a parse tree for a set of sentences, and then estimate the probabilities of each rule being used by counting them over the corpus. The MLE estimate for a rule $A \rightarrow \alpha$ is given by the expression

$$P_{MLE}(A \rightarrow \alpha) = \frac{\text{Count}(A \rightarrow \alpha)}{\sum_{\alpha} \text{Count}(A \rightarrow \alpha)}$$

If our training corpus consists of two parse trees (as shown in Figure 4.14), we will get the estimates as shown in Table 4.5 for the rules.

Table 4.5 The MLE for the grammar rules used in trees of Figure 4.14

Rule	Count (A → α)	Count A	MLE estimates
S → VP	2	2	1
NP → Det Noun PP	1	4	0.25
NP → Det Noun	3	4	0.75
VP → Verb NP	2	3	0.66
VP → VP PP	1	3	0.33
Det → the	2	2	1
Noun → hole	2	4	0.5
Noun → door	2	4	0.5
Prep → with	1	1	1
Verb → Paint	1	1	1

We now turn to another important question—what do we do with these probabilities? We assign a probability to each parse tree φ of a sentence s. The probability of a complete parse is calculated by multiplying the probabilities for each of the rules used in generating the parse tree:

$$P(\varphi, s) = \prod_{n \in \varphi} p\{r(n)\}$$

where n is a node in the parse tree φ and r is the rule used to expand n.

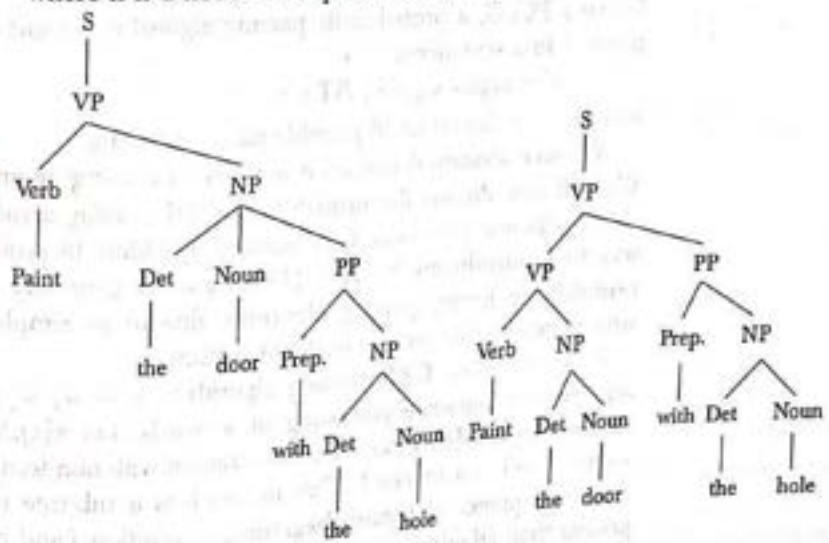


Figure 4.14 Two possible parse trees

For a sentence, the parse trees generated by a PCFG are the same as those generated by a corresponding CFG. However, the PCFG assigns a probability to each parse. The probability of the two parse trees of the sentence *Paint the door with the hole* (shown in Figure 4.14) can be computed as follows:

$$P(t1) = 0.2 * 0.5 * 0.2 * 0.2 * 0.35 * 0.25 * 1.0 * 0.25 * 0.4 * 0.35 * 0.25$$
$$= 0.0000030625$$
$$P(t2) = 0.2 * 0.2 * 0.5 * 0.2 * 0.4 * 0.35 * 0.25 * 1 * 0.25 * 0.4 * 0.35 * 0.25$$
$$= 0.000001225$$

In this example, the first tree has a higher probability leading to correct interpretation.

We can calculate probability to a sentence s by summing up probabilities of all possible parses associated with it.

$$P(s) = \sum_{t \in \tau(s)} P(t, s)$$
$$= \sum_{t \in \tau(s)} P(t)$$

Thus, the sentence will have the probability

$$P(t1) + P(t2) = 0.0000030625 + 0.000001225$$
$$= 0.0000042875$$

4.5.2 Parsing PCFGs

Given a PCFG, a probabilistic parsing algorithm assigns the most likely parse φ' to a sentence s.

$$\varphi' = \text{argmax}_{T \in \tau}(S)\ P(T \mid S)$$

where $\tau(S)$ is the set of all possible parse trees of s.

We have already discussed a number of parsing algorithms for CFG. We will now discuss the probabilistic CYK parsing algorithm. We have already discussed the basic CYK parsing algorithm. Its probabilistic version was first introduced by Ney (1991) and is generally preferred over probabilistic Earley parsing algorithm due to its simplicity. We begin with notations that we use in the algorithm.

As in the basic CYK parsing algorithm, $w = w_1\ w_2\ w_3\ w_i \ldots w_j \ldots w_n$ represents a sentence consisting of n words. Let $\varphi[i,j,A]$ represent the maximum probability parse for a constituent with non-terminal A spanning words i, $i+1$, up to $i+j-1$. This means it is a sub-tree rooted at A that derives sequence of j words beginning at position i and has a probability greater than all other possible sub-trees deriving the same word sequence.

An array named BP is used to store back pointers. These pointers allow us to recover the best parse. The output for a successful parse is the maximum probability parse $\varphi[1,n,S]$, which corresponds to a parse tree rooted at S and spanning the sequence of words $w_1\ w_2\ w_3 \ldots w_n$, i.e. the whole sentence.

```
Initialization:
  for i := 1 to n do
    for all rules A→ w_i do
      φ[i,1,A] = P(A → w_i)

Recursive Step:
  for j = 2 to n do
    for i = 1 to n-j+1 do
    begin
      φ[i,1,A] = φ
        for k = 1 to j-1 do
          φ'[i,j,A] = max_k φ'[i,k,B] × φ'[k,j,C] × P(A → BC),
          such that A → BC is a production rule in grammar
      BP[i,j,A] = { k, A, B}
    end
```

Figure 4.15 Probabilistic CYK algorithm

The algorithm is given in Figure 4.15. Like the basic CYK algorithm, the first step is to generate items of length 1. However, in this case, we have to initialize the maximum probable parse trees deriving a string of length 1, with the probabilities of the terminal derivation rules used to derive them.

$$\varphi[i,1,A] = P(A \rightarrow w_i)$$

Similarly, recursive step involves breaking a string into all possible ways and identifying the maximum probable parse.

$$\varphi'[i,j,A] = \max_k \varphi'[i,k,B] \times \varphi'[k,j,C] \times P(A \rightarrow BC)$$

The rest of the steps follow those of basic CYK parsing algorithm.

4.5.3 Problems with PCFG

The PCFG is not without disadvantages. Its first problem lies in the independence assumption. We calculate the probability of a parse tree assuming that the rules are independent of each other. However, this is not true. How a node expands depends on its location in the parse tree. For example, Francis et al. (1999) showed that pronouns occur more

frequently as subjects rather than objects. These dependencies are not captured by a PCFG, as the probability of, say, expanding an NP as a pronoun versus a lexical NP, is independent of whether the NP appears as a subject or an object.

Another problem associated with a PCFG is its lack of sensitivity to lexical information. Lexical information plays a major role in determining correct parse in case of PP attachment ambiguities and coordination ambiguities (Collins 1999). Two structurally different parses that use the same rules will have the same probability under a PCFG, making it difficult to identify the correct or most probable parse. The words appearing in a parse may make certain parses unnatural. This however, requires a model which captures lexical dependency statistics for different words. Such a model is presented next.

Lexicalization

In PCFG, the chance of a non-terminal expanding using a particular rule is independent of the actual words involved. However, this independence assumption does not seem reasonable. Words do affect the choice of the rule. Investigations made on tree bank data suggest that the probabilities of various common sub-categorization frames differ depending on the verb that heads the verb phrase (Manning and Schutze 1999). This suggests the need for lexicalization, i.e., involvement of actual words in the sentences, to decide the structure of the parse tree. Lexicalization is also helpful in choosing phrasal attachment positions. This model of lexicalization is based on the idea that these are strong lexical dependencies between heads and their dependents, for example, between a head noun and its modifiers, or between a verb and a noun phrase object, where the noun phrase object in turn can be approximated by its head noun.

One way to achieve lexicalization is to mark each phrasal node in a parse tree by its head word. Figure 4.16 is an example of such a tree. One way to implement this model is to use a lexicalized grammar. A lexicalized grammar is a grammar in which every finite structure is associated with one or more lexical heads. A context free grammar is not lexicalized as no lexical item is associated with its rule, such as, S → NP VP. Nor is the PCFG lexicalized. In order to convert a PCFG into lexicalized grammar, each of its rules must identify a head daughter, which is one of the constituents appearing on its right hand side, for example head daughter of S → NP VP rule is VP. The head word of a node in the parse tree is set to the head word of its head daughter. For example, the head of a verb phrase is the main verb. Hence, the head of the node VP in the Figure 4.16 is jumped. Similarly, the head of a constituent expanded using the rule S → NP VP is the head of VP.

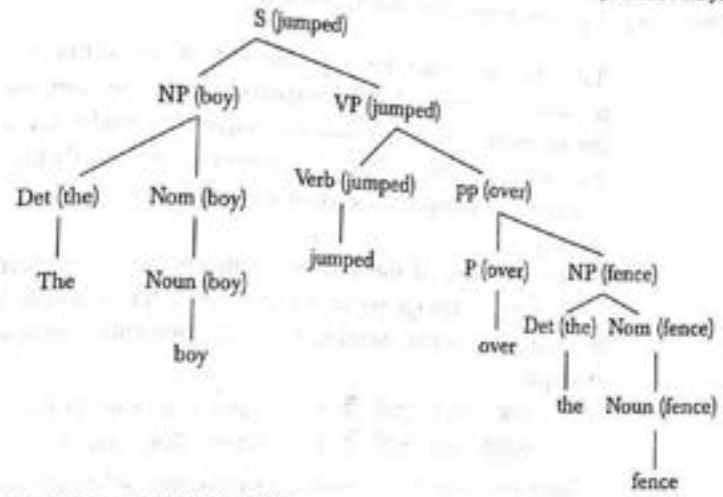


Figure 4.16 A lexicalized tree

The probability in a lexicalized PCFG is calculated for each rule or head of a phrase and head of the sub-phrase or head of the phrase (i.e., head/mother head) combination. For example, we need to collect the following rule/head probability:

VP (jumped) → verb (jumped) PP (over)
VP (jumped) → verb (jumped) PP (into)
VP (jumped) → verb (jumped) PP (to)

An example of the head or mother head combination is the following type of probability:

"What is the probability that an NP whose mother's head is *over* has the head *fence*?"

With a lexicalized PCFG, the probability of the parse tree is computed as the product of the probability of the head of the constituent c given the probability of the mother m(c) and the probability of the rule used to expand constituent c given the head of constituent (Charniak 1997):

$$P(\varphi, s) = \prod_{c \in \varphi} p[(h(c)|m(c)] \cdot p[(r(c)|h(c)]$$

where h(c) = head of the constituent c
r(c) = rule used to expand constituent c
and m(c) = mother of the constituent c

4.6 INDIAN LANGUAGES

Not all natural languages have the same characteristics. So far we have talked only of the English language, using CFG as the grammar formalism.

But CFG may not be a viable choice for other languages. This is particularly true for Indian languages. In this section, we discuss some of the characteristics of Indian languages that make CFG unsuitable. We then give a brief outline of a parser-based on Paninian grammar. As discussed in Chapter 2, Paninian grammar can be used to model Indian languages.

The majority of the Indian languages are free word order. By free word order language we mean that the order of words in a sentence can be changed without leading to a grammatically incorrect sentence. For example:

सबा खाना खाती है । *Saba khana khati hai.*
खाना सबा खाती है । *Khana Saba khati hai.*

Both are valid Hindi sentences meaning *Saba eats food.*

The same is true for Urdu and most other Indian languages. The CFG we used for parsing English is basically positional. It can be used to model language in which a position of the constituents carries useful information, but it fails to model free word order languages.

Extensive and productive use of complex predicates (CPs) is another property that most Indian languages have in common. A complex predicate combines a light verb with a verb, noun, or adjective, to produce a new verb. For example,

(a) सबा आयी ।
(*Saba Ayi.*)
Saba came.

(b) सबा आ गयी ।
(*Saba a gayi.*)
Saba come went.
Saba arrived.

(c) सबा आ पडी ।
Saba a pari.
Saba come fell.
Saba came (suddenly).

The complex predicates change the functional structure of the sentence; however they do not change the sub-categorization frame of the verb.

The use of post-position case markers (Vibhakti) and verb complexes consisting of sequences of verbs, e.g., खा रही है (*kha rahi hai*) are other properties common to Indian languages. The auxiliary verbs in this sequence provide information about tense, aspect, and modality. Paninian grammar provides a framework to model Indian languages. It focuses on the extraction of Karak relations from a sentence.

Bharti and Sangal (1990) described an approach for parsing of Indian languages based on Paninian grammar formalism. Their parser works in two stages. The first stage is responsible for identifying word groups and the second for assigning a parse structure to the input sentence. The input to the first stage comes from the morphological analysis phases (as discussed in Chapter 3). The output of the first stage is word groups, which are sequences of words that act as a unit. For example, a verb group consists of a main verb and a sequence of auxiliaries. For the sentence

लडकियाँ मैदान में हाकी खेल रही हैं । (4.8)
Ladkiyan maidaan mein hockey khel rahi hein.

The word *ladkiyan* forms one unit, the words *maidaan* and *mein* are grouped together to form a noun group, and the word sequence *khel rahi hein* forms a verb group.

The choice of noun and verb groups over noun and verb phrases adds computational simplicity to the approach. The concept of verb phrase is not natural to Indian languages and computing a noun phrase is difficult.

In the second stage, the parser takes the word groups formed during first stage and identifies (i) Karaka relations among them, and (ii) senses of words. A data structure, called Karaka chart, stores additional information like Karaka–Vibhakti mapping, Karaka necessity (mandatory or optional), and transformation rules for Karaka relations, needed in this step. Transformation rules tell us how to create a Karaka chart for a verb group using the default Karaka chart. The form of the default Karaka chart is shown in Table 4.6.

Table 4.6 Default Karaka chart

Karaka (case relations)	Vibhakti (case markers or post-positions)	Necessity
Karta	ϕ	Mandatory
Karma	*Ko* or ϕ	Mandatory
Adhikaran	*Mein* or *par*	Optional
Sampradan	*Ko* or *ke liye*	Optional

Once the Karaka chart for the verb groups are ready, noun groups are tested against them. A noun group satisfying the Vibhakti restriction for a verb group becomes a candidate for its Karaka. The Karaka relation between a verb group and a noun group can be depicted using a constraint graph. Nodes in the graph represent word groups and an arc represents a Karaka restriction between word groups. We have earlier identified the word groups in sentence (4.8). Its constraint graph is given in Figure 4.17.

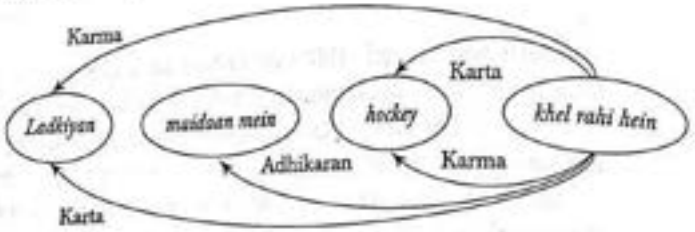


Figure 4.17 Constraint graph for sentence (4.8)

Each sub-graph of the constraint graph that satisfies the following constraints yields a parse of the sentence.

1. It contains all the nodes of the graph.
2. It contains exactly one outgoing edge from a verb group for each of its mandatory Karakas. These edges are labelled by the corresponding Karaka.
3. For each of the optional Karaka in Karaka chart, the sub-graph can have at most one outgoing edge labelled by the Karaka from the verb group.
4. For each noun group, the sub-graph should have exactly one incoming edge.

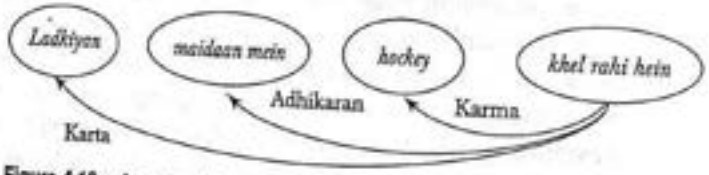


Figure 4.18 A parse of the sentence (4.8)

A sub-graph of the constraint graph (Figure 4.17) satisfying these constraints is shown in Figure 4.18. This represents one of the possible parse of sentence (4.8). More than one sub-graph may satisfy these constraints in which case the sentence is ambiguous and has multiple parse associated with it. If no sub-graph satisfies these constraints, then the grammar fails to assign any parse structure to the sentence. Disambiguation among various senses of verbs and nouns is carried out with the help of the lakshan chart, which contains features that are useful for discriminating among various senses. Readers are referred to Bharti and Sangal (1990) and Bharti et al. (1995) for detailed treatment of the algorithm.

SUMMARY

This chapter has introduced a number of important notions about syntax.

- Syntactic parsing deals with the syntactic structure of a sentence.
- In many languages, words are grouped to form larger groups termed constituent or phrases, which can be modelled by context-free grammar (CFG).
- A CFG consists of a set of rules or productions stating which elements can occur in a phrase and in which order.
- The set of all the strings containing terminal symbols that can be derived from the start symbol of the grammar, defines the language generated by that grammar.
- Phrase types are named after their head, which is the important lexical category that determines the properties of a phrase. If the head is a noun the phrase is called noun phrase, if head is a verb, the phrase is called a verb phrase, and so forth.
- Grammatical categories impose certain constraints: subject-verb agreement is one of them. The subjects in a sentence must agree with the main verb in number and person. Grammar rules should be able to ensure this agreement.
- CFG can't handle subject-verb agreement efficiently. Feature structures can be used to efficiently capture the properties of grammatical categories.
- Top-down and bottom-up are two commonly used parsing approaches.
- The Earley parser and the CYK parser are dynamic programming algorithms for parsing. The Earley algorithm takes a top-down approach whereas the CYK takes a bottom-up approach.
- Probabilistic parsing assigns the most likely parse to a sentence and hence, can be used to handle parsing ambiguities. However, it is insensitive to lexical information.
- Indian languages are free word order languages and cannot be modelled by CFG.

REFERENCES

Backus, J.W., 1959, 'The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM conference,' *Proceedings of the International Conference on Information Processing*, UNESCO, pp. 125–32.

CHAPTER 9

INFORMATION RETRIEVAL–1

CHAPTER OVERVIEW

The huge amount of information stored in electronic form, has placed heavy demands on information retrieval systems. This has made information retrieval an important research area. This chapter is concerned with the design of information retrieval systems. It discusses design features of systems and introduces various models, such as the classical (boolean, probabilistic, and vector space retrieval models), non-classical (information logic, situation theory, and interaction information retrieval model), and alternative information retrieval (cluster, fuzzy, and LSI) models. A detailed discussion of vector space model is also given. The final topic of discussion in this chapter is information retrieval evaluation models.

9.1 INTRODUCTION

Information retrieval (IR) deals with the organization, storage, retrieval, and evaluation of information relevant to a user's query. A user in need of information formulates a request in the form of a query written in a natural language. The retrieval system responds by retrieving the document that seems relevant to the query. Research in IR is not new. It dates back to the 1960s when text retrieval systems were introduced. Traditionally, however, it is not considered an important application area of NLP. Interest in the field was generated due to the insurgence of the World Wide Web. The interaction between NLP and IR is now strengthening. Many NLP techniques, including the probabilistic model, have found application in IR systems and techniques such as latent semantic indexing, vector space retrieval, etc.

Traditionally, IR systems are not expected to return the actual information, only documents containing that information. As Lancaster (1979) pointed out:

An information retrieval system does not inform (i.e., change the knowledge of the user on the subject of her inquiry. It merely informs on the existence (or non-existence) and whereabouts of documents relating to her request.

The word *document* is a general term that includes non-textual information such as images and speech.

Our concern in this chapter is only with retrieval of text documents. This excludes question answering systems and data retrieval systems. The three systems differ in the nature of their queries and expected results of the queries. In both question answering systems and data retrieval systems, queries are very specific and precise in nature. On the other hand, queries submitted to IR systems are often vague and imprecise. A question answering system provides users with the answers to specific questions. Data retrieval systems retrieve precise data, usually organized in a well-defined structure. Unlike data retrieval systems, IR systems do not search for specific data. Nor do they search for direct answers to question, as question answering systems do.

9.2 DESIGN FEATURES OF INFORMATION RETRIEVAL SYSTEMS

Figure 9.1 illustrates the basic process of IR. It begins with the user's information need. Based on this need, he/she formulates a query. The IR system returns documents that seem relevant to the query. This is an engineering account of the IR system. The basic question involved is, 'what constitutes the information in the documents and the queries'. This, in turn is related to the problem of representation of documents and queries. The retrieval is performed by matching the query representation with document representation.

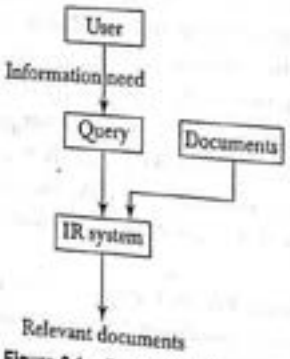


Figure 9.1 Basic information retrieval process

The actual text of the document is not used in the retrieval process. Instead, documents in a collection are frequently represented through a set of index terms or keywords. In this chapter, the word *term* and *keyword* will be used independently. Keywords can be single word or multi-word phrases. They might be extracted automatically or manually (i.e., specified by a human). Such a representation provides a logical view of the document. The process of transforming document text to some representation of it is known as *indexing*. There are different types of index structures. One used data structure, commonly by the IR system, is the inverted index. An inverted index is simply a list of keywords, with each keyword carrying pointers to the documents containing that keyword. The computational cost involved in adopting a full text logical view (i.e., using a full set of words to represent a document) is high. Hence, some text operations are usually performed to reduce the set of representative keywords. The two most commonly used text operations are *stop word elimination* and *stemming*. Stop word elimination removes grammatical or functional words, while stemming reduces words to their common grammatical roots. *Zipf's law* can be applied to further reduce the size of index set. Not all the terms in a document are equally relevant. Some might be more important in conveying a document's content. Attempts have been made to quantify the significance of index terms to a document by assigning them numerical values, called weights. The choice of index terms and weights is a difficult theoretical and practical problem and several technique are used to cope with it. A number of *term-weighting* schemes have been proposed in the literature over the years. We discuss a few of them in this chapter.

9.2.1 Indexing

In a small collection of documents, an IR system can access a document to decide its relevence to a query. However, in a large collection of documents, this technique poses practical problems. Hence, a collection of raw documents is usually transformed into an easily accessible representation. This process is known as indexing. Most indexing techniques involve identifying good document descriptors, such as keywords or terms, which describe the information content of documents. A good descriptor is one that helps describe the content of the document and discriminate it from other documents in the collection. Luhn (1957, 1958) is considered the first person to advance the notion of automatic indexing of documents based on their content. He assumed that the frequency of certain word-occurrences in an article gave meaningful

identification of the article's content. He proposed that the discrimination power of index terms is a function of the rank order of the frequency of their occurrence, and that middle frequency terms have the highest discrimination power. This model was proposed for the extraction of salient terms from a document. These extracted terms are then used to represent text. Thus, indexing is simply the representation of text (query and document) as a set of terms whose meaning is equivalent to some content of the original text.

As mentioned earlier, the word *term* can be a single word or multi-word phrases. For example, the sentence, *Design features of information retrieval systems*, can be represented as follows:

Design, features, information, retrieval, systems

It can also be represented by the set of terms:

Design, features, information retrieval, information retrieval systems

These multi-word terms can be obtained by looking at frequently appearing sequences of words, n-grams, part-of-speech tags, or by applying NLP to identify meaningful phrases or handcrafting. POS tagging helps extract meaningful sequences of words; it handles sense ambiguity, as words are assigned POS based on their local (sentential) context. Though statistical approaches to phrase extraction are more efficient, they fail to handle word order changes and structural variations, which are better handled by syntactic approaches. In text retrieval conference (TREC), the method used for phrase extraction is as follows:

1. Any pair of adjacent non-stop words is regarded a potential phrase.
2. The final list of phrases is composed of those pairs of words that occur in, say, 25 or more documents in the document collection.

The NLP is also used in the recognition of proper nouns and the normalization of noun phrases. Ideally, all names in the text need to be recognized and represented as a single entity, e.g. *President Kalam, President of India*, and variants of the same name are recognized as such. Phrase normalization captures structural variations in phrases. For example, the three phrases text categorization, categorization of text, and categorize text, are normalized to give text categorize.

9.2.2 Eliminating Stop Words

The lexical processing of index terms involves elimination of stop words. Stop words are high frequency words which have little semantic weight and are thus, unlikely to help in retrieval. These words play important

grammatical roles in language, such as in the formation of phrases, but do not contribute to the semantic content of a document in a keyword-based representation. Such words are commonly used in documents, regardless of topics, and thus, have no topical specificity. Typical example of stop words are articles and prepositions. Eliminating them considerably reduces the number of index terms. The drawback of eliminating stop words is that it can sometimes result in the elimination of useful index terms, for instance the stop word *A* in *Vitamin A*. Some phrases, like *to be or not to be*, consist entirely of stop words. Eliminating stop words in such case, make it impossible to correctly search a document. Table 9.1 lists some of the stop words in English.

Table 9.1 Sample stop words in English

about	above	accordingly	across	after
afterwards	again	against	all	almost
alone	along	already	also	although
am	among	amongst	always	an
and	another	any	anybody	anyhow
anyone	anything	anywhere	apart	are
around	as	aside	at	away
awfully	be	because	been	before
beforehand	behind	being	below	beside
besides	best	better	between	beyond
both	brief	but	by	can
could	did	during	each	even
everybody	everyone	everything	everywhere	else
even	ever	every	for	from
further	had	has	have	her
herself	him	himself	he	furthermore
many	near	shall	she	self
whose	why	will	where	ex
except	far	first	five	former
formerly	over	overall	usually	appropriate

9.2.3 Stemming

Stemming normalizes morphological variants, though in a crude manner, by removing affixes from the words to reduce them to their stem, e.g., the words *compute, computing, computes*, and *computer*, are all be reduced to same word stem, *comput*. Thus, the keywords or terms used to represent text are stems, not the actual words. One of the most widely used stemming

algorithms has been developed by Porter (1980). The stemmed representation of the text, *Design features of information retrieval systems*, is

(design, featur, inform, retriev, system)

Note that stop words have been removed in this representation and the remaining terms are in lower case.

One of the problems associated with stemming is that it may throw away useful distinctions. In some cases, it may be useful to help conflate similar terms, resulting in increased *recall*. In others, it may be harmful, resulting in reduced *precision* (e.g., when documents containing the term *computation* are returned in response to the query phrase *personal computer*). *Recall* and *precision* are the two most commonly used measures of the effectiveness of an information retrieval system, and are explained in detail later in this chapter.

9.2.4 Zipf's Law

Zipf made an important observation on the distribution of words in natural languages. This observation has been named Zipf's law. Simply stated, Zipf's law says that the frequency of words multiplied by their ranks in a large corpus is more or less constant. More formally,

Frequent × rank = constant

This means that if we compute the frequencies of the words in a corpus, and arrange them in decreasing order of frequency, then the product of the frequency of a word and its rank is approximately equal to the product of the frequency and rank of another word. This indicates that the frequency of a word is inversely proportional to its rank. This relationship is shown in Figure 9.2.

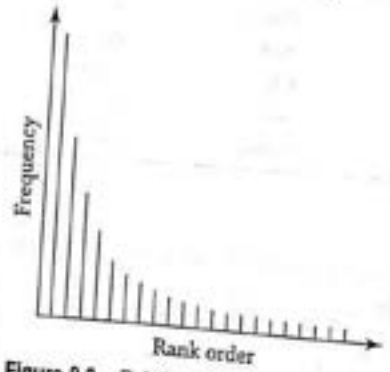


Figure 9.2 Relationship between the frequency of words and their rank order

Empirical investigation of Zipf's law on large corpuses suggest that human languages contain a small number of words that occur with high frequency and a large number of words that occur with low frequency. In between, is a middling number of medium frequency terms. This distribution has important significance in IR. The high frequency words being common, have less discriminating power, and thus, are not useful for indexing. Low frequency words are less likely to be included in the query, and are also not useful for indexing. As there are a large number of rare (low frequency) words, dropping them considerably reduces the size of a list of index terms. The remaining medium frequency words are content-bearing terms and can be used for indexing. This can be implemented by defining thresholds for high and low frequency, and dropping words that have frequencies above or below these thresholds. Stop word elimination can be thought of as an implementation of Zipf's law, where high frequency terms are dropped from a set of index terms.

9.3 INFORMATION RETRIEVAL MODELS

An IR model is a pattern that defines several aspects of the retrieval procedure, for example, how documents and user's queries are represented, how a system retrieves relevant documents according to users' queries, and how retrieved documents are ranked. The IR system consists of a model for documents, a model for queries, and a matching function which compares queries to documents. The central objective of the model is to retrieve all documents relevant to a query. This defines the central task of an IR system.

Several different IR models have been developed. These models differ in the way documents and queries are represented and retrieval is performed. Some of them consider documents as sets of terms and perform retrieval based merely on the presence or absence of one or more query terms in the document. Others represent a document as a vector of term weights and perform retrieval based on the numeric score assigned to each document, representing similarity between the query and the document. These models can be classified as follows:

- Classical models of IR
- Non-classical models of IR
- Alternative models of IR

The three classical IR models—Boolean, vector, and probabilistic—are based on mathematical knowledge that is easily recognized and well understood. These models are simple, efficient, and easy to implement.

Almost all existing commercial systems are based on the mathematical models of IR. That is why they are called classical models of IR.

Non-classical models perform retrieval based on principles other than those used by classical models, i.e., similarity, probability, and Boolean operation. These are best exemplified by models based on special logic technique, situation theory, or the concept of interaction.

The third category of IR models, namely alternative models, are actually enhancements of classical models, making use of specific techniques from other fields. The cluster model, fuzzy model, and latent semantic indexing (LSI) model are examples of alternative models of IR.

9.4 CLASSICAL INFORMATION RETRIEVAL MODELS

9.4.1 Boolean model

Introduced in the 50s, the Boolean model is the oldest of the three classical models. It is based on Boolean logic and classical set theory. In this model, documents are represented as a set of keywords, usually stored in an inverted file. An inverted file is a list of keywords and identifiers of the documents in which they occur. Users are required to express their queries as a Boolean expression consisting of keywords connected with Boolean logical operators (AND, OR, NOT). Retrieval is performed based on whether or not document contains the query terms.

Given a finite set

$$T = \{t_1, t_2, \ldots, t_i, \ldots, t_m\}$$

of index terms, a finite set

$$D = \{d_1, d_2, \ldots, d_j, \ldots, d_n\}$$

of documents and a Boolean expression—in a normal form—represent a query Q as follows:

$$Q = \wedge\{\vee\theta\}, \theta_i \in \{t_i, \neg t_i\}$$

The retrieval is performed in two steps:

1. The set R_i of documents are obtained that contain or do not contain the term t_i:

   $$R_i = \{d_j \mid \theta_i \in d_j\}, \theta_i \in \{t_i, \in t_i\}$$

   where $\neg t_i \in d_j$ means $t_i \in d_j$

2. Set operations are used to retrieve documents in response to Q:

   $$\cap R_i$$

Example 9.1 Let the set of original documents be

$$D = \{D_1, D_2, D_3\}$$

where

D_1 = Information retrieval is concerned with the organization, storage, retrieval, and evaluation of information relevant to user's query.

D_2 = A user having an information needs to formulate a request in the form of query written in natural language.

D_3 = The retrieval system responds by retrieving the document that seems relevant to the query.

Let the set of terms used to represent these documents be

$$T = \{\text{information, retrieval, query}\}$$

Then, the set D of document will be represented as follows:

$$D = \{d_1, d_2, d_3\}$$

where

$$d_1 = \{\text{information, retrieval, query}\}$$
$$d_2 = \{\text{information, query}\}$$
$$d_3 = \{\text{retrieval, query}\}$$

Let the query Q be

$$Q = \text{information} \wedge \text{retrieval}$$

First, the sets R_1 and R_2 of documents are retrieved in response to Q, where

$$R_1 = \{d_j \mid \text{information} \in d_j\} = \{d_1, d_2\}$$
$$R_2 = \{d_j \mid \text{retrieval} \in d_j\} = \{d_1, d_3\}$$

Then, the following documents are retrieved in response to query Q

$$\{d_j \mid d_j \in R_1 \cap R_2\} = \{d_1\}$$

This results in the retrieval of the original document D_1 that has the representation d_1. If more than one document have the same representation, every such document is retrieved. Boolean information retrieval does not differentiate between these documents. With an inverted index, this simply means taking an intersection of the list of the documents associated with the keywords *information* and *retrieval*.

Boolean retrieval models have been used in IR systems for a long time. They are simple, efficient, and easy to implement and perform well in terms of recall and precision if the query is well formulated. However, the model suffers from certain drawbacks.

First, the model is not able to retrieve documents that are only partly relevant to user query; all information is 'to be or not to be'.

Second, a Boolean system is not able to rank the returned list of documents. It distinguishes between presence and absence of keywords but fails to assign relevance and importance to keywords in a document.

Third, users seldom formulate their query in the pure Boolean expression that this model requires.

Numerous extensions of the Boolean model have been suggested to overcome these weaknesses. The best of these are the P-norm model developed by Salton et al. (1983) and a fuzzy-set model suggested by Paice (1984).

9.4.2 Probabilistic Model

The probabilistic model applies a probabilistic framework to IR. It ranks documents based on the probability of their relevance to a given query (Robertson and Jones 1976). Retrieval depends on whether probability of relevance (relative to a query) of a document is higher than that of non-relevance, i.e. whether it exceeds a threshold value. Given a set of documents D, a query q, and a cut-off value α, this model first calculates the probability of relevance and irrelevance of a document to the query. It then ranks documents having probabilities of relevance at least that of irrelevance in decreasing order of their relevance. Documents are retrieved if the probability of relevance in the ranked list exceeds the cut off value.

More formally, if $P(R/d)$ is the probability of relevance of a document d, for query q, and $P(I/d)$ is the probability of irrelevance, then the set of documents retrieved in response to the query q is as follows.

$$S = \{d_j \mid P(R/d_j) \geq P(I/d_j)\}\ P(R/d_j) \geq \alpha$$

Development of the probabilistic model was carried out largely by Maron and Kuhns (1960), Robertson and Sparck-Jones (1976), Robertson et al. (1982). Different mathematical methods for calculating the probabilities of relevance and irrelevance, as well as properties and applications, are discussed by Van Rijsbergen (1992), Callan et al. (1992), and Fur (1992).

Most of the systems assume that terms are independent when estimating probabilities for the probabilistic model. This assumption allows for accurate estimation of parameter values and helps reduce computational complexity of the model. However, this assumption seems to be inaccurate, as terms in a given domain usually tend to co-occur. For example, it is more likely that 'match point' will co-occur with 'tennis' rather than 'cricket'. Different forms of assumption are discussed in Robertson (1977). A comparison of the Boolean and probabilistic models can be found in Losee (1997). A comprehensive review of the probabilistic model is presented by Crestani et al. (1998). A baysian network version of the probabilistic model forms the basis of the INQUERY system (Callan et al. 1992).

The probabilistic model, like the vector model, can produce results that partly match the user query. Nevertheless, this model has drawbacks, one of which is the determination of a threshold value for the initially retrieved set; the number of relevant documents by a query is usually too small for the probability to be estimated accurately.

9.4.3 Vector Space Model

The vector space model is one of the most well-studied retrieval models. Important contribution to its development was made by Luhn (1959), Salton (1968), Salton and McGill (1983), and van Rijsbergen (1977). The vector space model represents documents and queries as vectors of features representing terms that occur within them. Each document is characterized by a Boolean or numerical vector. These vectors are represented in a multi-dimensional space, in which each dimension corresponds to a distinct term in the corpus of documents. In its simplest form, each feature takes a value of either zero or one, indicating the absence or presence of that term in a document or query. More generally, features are assigned numerical values that are usually a function of the frequency of terms. Ranking algorithms compute the similarity between document and query vectors, to yield a retrieval score to each document. This score is used to produce a ranked list of retrieved documents. Given a finite set of n documents

$$D = \{d_1, d_2, \ldots, d_j, \ldots, d_n\}$$

and a finite set of m terms

$$T = \{t_1, t_2, \ldots, t_i, \ldots, t_m\}$$

each document is represented by a column vector of weights as follows:

$$(w_{1j}, w_{2j}, w_{3j}, \ldots, w_{ij}, \ldots, w_{mj})^t$$

where w_{ij} is the weight of the term t_i in document d_j. The document collection as a whole is represented by an $m \times n$ term-document matrix as

$$\begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1j} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2j} & \cdots & w_{2n} \\ w_{i1} & w_{i2} & \cdots & w_{ij} & \cdots & w_{in} \\ w_{m1} & w_{m2} & \cdots & w_{mj} & \cdots & w_{mn} \end{pmatrix}$$

Different term-weighting functions have been introduced (Salton and Buckley 1988). We discuss some of them in this section.

Third, users seldom formulate their query in the pure Boolean expression that this model requires.

Numerous extensions of the Boolean model have been suggested to overcome these weaknesses. The best of these are the P-norm model developed by Salton et al. (1983) and a fuzzy-set model suggested by Paice (1984).

9.4.2 Probabilistic Model

The probabilistic model applies a probabilistic framework to IR. It ranks documents based on the probability of their relevance to a given query (Robertson and Jones 1976). Retrieval depends on whether probability of relevance (relative to a query) of a document is higher than that of non-relevance, i.e. whether it exceeds a threshold value. Given a set of documents D, a query q, and a cut-off value α, this model first calculates the probability of relevance and irrelevance of a document to the query. It then ranks documents having probabilities of relevance at least that of irrelevance in decreasing order of their relevance. Documents are retrieved if the probability of relevance in the ranked list exceeds the cut off value.

More formally, if $P(R/d)$ is the probability of relevance of a document d, for query q, and $P(I/d)$ is the probability of irrelevance, then the set of documents retrieved in response to the query q is as follows.

$$S = \{d_j \mid P(R/d_j) \geq P(I/d_j)\}\ P(R/d_j) \geq \alpha$$

Development of the probabilistic model was carried out largely by Maron and Kuhns (1960), Robertson and Sparck-Jones (1976), Robertson et al. (1982). Different mathematical methods for calculating the probabilities of relevance and irrelevance, as well as properties and applications, are discussed by Van Rijsbergen (1992), Callan et al. (1992), and Fur (1992).

Most of the systems assume that terms are independent when estimating probabilities for the probabilistic model. This assumption allows for accurate estimation of parameter values and helps reduce computational complexity of the model. However, this assumption seems to be inaccurate, as terms in a given domain usually tend to co-occur. For example, it is more likely that 'match point' will co-occur with 'tennis' rather than 'cricket'. Different forms of assumption are discussed in Robertson (1977). A comparison of the Boolean and probabilistic models can be found in Losee (1997). A comprehensive review of the probabilistic model is presented by Crestani et al. (1998). A baysian network version of the probabilistic model forms the basis of the INQUERY system (Callan et al. 1992).

The probabilistic model, like the vector model, can produce results that partly match the user query. Nevertheless, this model has drawbacks, one of which is the determination of a threshold value for the initially retrieved set; the number of relevant documents by a query is usually too small for the probability to be estimated accurately.

9.4.3 Vector Space Model

The vector space model is one of the most well-studied retrieval models. Important contribution to its development was made by Luhn (1959), Salton (1968), Salton and McGill (1983), and van Rijsbergen (1977). The vector space model represents documents and queries as vectors of features representing terms that occur within them. Each document is characterized by a Boolean or numerical vector. These vectors are represented in a multi-dimensional space, in which each dimension corresponds to a distinct term in the corpus of documents. In its simplest form, each feature takes a value of either zero or one, indicating the absence or presence of that term in a document or query. More generally, features are assigned numerical values that are usually a function of the frequency of terms. Ranking algorithms compute the similarity between document and query vectors, to yield a retrieval score to each document. This score is used to produce a ranked list of retrieved documents. Given a finite set of n documents

$$D = \{d_1, d_2, \ldots, d_j, \ldots, d_n\}$$

and a finite set of m terms

$$T = \{t_1, t_2, \ldots, t_i, \ldots, t_m\}$$

each document is represented by a column vector of weights as follows:

$$(w_{1j}, w_{2j}, w_{3j}, \ldots, w_{ij}, \ldots, w_{mj})^t$$

where w_{ij} is the weight of the term t_i in document d_j. The document collection as a whole is represented by an $m \times n$ term-document matrix as

$$\begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1j} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2j} & \cdots & w_{2n} \\ w_{i1} & w_{i2} & \cdots & w_{ij} & \cdots & w_{in} \\ w_{m1} & w_{m2} & \cdots & w_{mj} & \cdots & w_{mn} \end{pmatrix}$$

Different term-weighting functions have been introduced (Salton and Buckley 1988). We discuss some of them in this section.

Example 9.2 Consider the documents and terms in Example 9.1. Let the weights be assigned based on the frequency of the term within the document. Then, the associated vectors will be

(2, 2, 1)
(1, 0, 1)
(0, 1, 1)

The vectors can be represented as a point in Euclidean space, where the coordinates of point P_j are the components of d_j. The term-document matrix is

$$\begin{pmatrix} 2 & 1 & 0 \\ 2 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

This raw term frequency approach gives too much importance to the absolute values of various coordinates of each document. For example, a document with weights 4, 4, and 2, would be quite similar to document 1, except for the differences in magnitude of term weights.

To reduce the importance of the length of document vectors, we normalize document vectors. Normalization changes all vectors to a standard length. We convert document vectors to unit length by dividing each dimension by the overall length of the vector. Normalizing the term-document matrix shown in this example, we get the following matrix

$$\begin{pmatrix} 0.67 & 0.71 & 0 \\ 0.67 & 0 & 0.71 \\ 0.33 & 0.71 & 0.71 \end{pmatrix}$$

Elements of each column are divided by the length of the column vector given by $\sqrt{\sum_i w_{ij}^2}$. The values shown in this matrix have been rounded to two decimal digits.

9.4.4 Term Weighting

Each term that is selected as an indexing feature for a document, acts as a discriminator between that document and all other documents in the corpus. Luhn (1958) attempted to quantify the discriminating power of the terms by associating the frequency of their occurrence (term frequency) within the document. He postulated that the most discriminating (concept bearing) terms are mid frequency terms. This postulate can be refined by noting the following facts:

1. The more a document contains a given word, the more that document is about a concept represented by that word.
2. The less a term occurs in particular document in a collection, the more discriminating that term is.

The first factor simply means that terms that occur more frequently represent the document's meaning more strongly than those occurring less frequently, and hence should be given high weights. In the simplest form, this weight is the raw frequency of the term in the document, as discussed earlier. The second factor actually considers term distribution across the document collection. Terms occurring in a few documents are useful for distinguishing those documents from the rest of the collection. Similarly, terms that occur more frequently across the entire collection are less helpful while discriminating among documents. This requires a measure that favours terms appearing in fewer documents. The fraction n/n_i—where n is the total number of the document in the collection and n_i the number of the document in which the term i occurs—exactly gives this measure. This measure assigns the lowest weight 1 to a term that appears in all documents and the highest weight of n to a term that occurs in only one document. As the number of documents in any collection is usually large, the log of this measure is usually taken, resulting in the following form of inverse document frequency (idf) term weight:

$$\text{idfi} = \log\left(\frac{n}{n_i}\right)$$

Inverse document frequency (idf) attaches more importance to more specific terms. If a term occurs in all documents in a collection, its idf is 0. Researchers have attempted to include term distribution in the weighting function (Sparck-Jones 1972, Salton 1971) to give a more accurate quantification of term importance. Sparck-Jones showed experimentally that a weight of $\log(n/n_i)+1$, termed as inverse document frequency, leads to more effective retrieval. Later researchers attempted to combine term frequency (tf) and idf weights, resulting in a family of tf × idf weight schemes having the following general form:

$$w_{ij} = \text{tf}_{ij} \times \log\left(\frac{n}{n_i}\right)$$

According to this scheme, the weight of a term t_i in document d_j is equal to the product of document frequency of the term and log of its inverse document frequency within the collection. The tf × idf weighting scheme combines both the 'local' and 'global' statistics to assign term weight. The tf component is a document specific statistic that measures

the importance of a term within the document, whereas the idf is a global statistic that attempts to include distribution of the term across the document collection. These weighting schemes are well suited to the ad-hoc retrieval environment but pose problems in routing environment, where no fixed document collection exists. Usually, a training set of documents is used to compute statistics in such an environment and it is assumed that subsequent documents arriving at the system have the same statistical properties as the training set. We now give an example to explain how term weights can be calculated using the tf–idf weighting scheme.

Example 9.3 Consider a document represented by the three terms {tornado, swirl, wind} with the raw tf 4, 1, and 1 respectively. In a collection of 100 documents, 15 documents contain the term *tornado*, 20 contain *swirl*, and 40 contain *wind*. The idf of the term *tornado* can be computed as

$$\log\left(\frac{n}{n_i}\right) = \log\left(\frac{100}{15}\right) = 0.82$$

The idf of other terms are computed in the same way. Table 9.2 shows the weights assigned to the three terms using this approach.

Table 9.2 Computing tf-idf weight

Term	Frequency (tf)	Document frequency (n_i)	idf [$\log(n/n_i)$]	Weight (tf × idf)
Tornado	4	15	0.824	0.296
Swirl	1	20	0.699	0.699
Wind	1	40	0.398	0.389

Many variations of tf × idf measure have been reported. Some of these attempt to normalize tf and idf factors in different ways to allow for variations in document length (Salton and Buckley 1988). One way to normalize tf is to divide it by the frequency of the most frequent term in the document. This kind of normalization, often termed as maximum normalization, yields a value between 0 and 1. Normalization is needed because using absolute (raw) term frequency to weight terms favours longer documents over shorter ones. After frequency normalization, the weight of a term in a given document depends on the frequency of its occurrence in relation to the other terms in the same document, instead of its absolute frequency. Similarly, idf can be normalized by dividing it by the logrithm of the collection size (n).

$$w_{ij} = \frac{\text{tf}_{ij}}{\max(\text{tf}_{ij})} \times \log\left(\frac{n}{n_i}\right) / \log n$$

A third factor that may affect weighting function is the document length. A term appearing the same number of times in a short document and in a long document, will be more valuable to the former. Most weighting schemes can thus be characterized by the following three factors:

- Within-document frequency or term frequency (tf)
- Collection frequency or inverse document frequency (idf)
- Document length

Any term weighting scheme can be represented by a triple ABC. The letter A in this triple represents the way the tf component is handled, *B* indicates the way the idf component is incorporated, and *C* represents the length normalization component. Possible options for each of the three dimensions of the triple are shown in Table 9.3. Different combinations of options can be used to represent document and query vectors. The retrieval model themselves can be represented by a pair of triples like nnn.nnn (doc = 'nnn', query = 'nnn'), where the first triple corresponds to the weighting strategy used for the documents and the second triple to the weighting strategy used for the query term.

Table 9.3 Calculating weight with different options for the three weighting factors

	Term frequency within document		
A	*n*	$\text{tf} = \text{tf}_{ij}$	Raw term frequency
	b	tf = 0 or 1 (binary weight)	
	a	$\text{tf} = 0.5 + 0.5\left(\frac{\text{tf}_{ij}}{\max \text{tf in } D_j}\right)$	Augmented term frequency
	l	$\text{tf} = \ln(\text{tf}_{ij}) + 1.0$	Logarithmic term frequency
	L	$\text{tf} = \frac{\ln(\text{tf}_{ij} + 1.0)}{1.0 + \ln[\text{mean}(\text{tf in } D_j)]}$	Average term frequency-based normalization
B	**Inverse document frequency**		
	n	wt = tf	No conversion
	t	$\text{wt} = \text{tf} \cdot \ln\left(\frac{n}{n_i}\right)$	Multiply tf with idf
C	**Document length**		(no conversion)
	n	$w_{ij} = \text{wt}$	
	c	w_{ij} is obtained by dividing each wt by sqrt [sum of (wts squared)]	

There are many ways to compute each component. The simplest is to use either binary weight or raw term frequency. As shown in Table 9.3, the first occurrence of a term is more important than successive repeating occurrences. Thus, tf can be computed as $0.5 + 0.5$ (tf_{ij}/max tf in d_j) in which normalization is achieved by dividing tf by maximum tf value for any term in the document, or as $\ln(tf_{ij}) + 1.0$, which is known as logarithmic term frequency. The former computation is called augmented normalized term frequency. It causes tf to vary between 0.5 and 1. The problem with maximum normalization and augmented normalization of the tf component is that a single term in a document, with an unusually high frequency, may degrade the weights of the other terms significantly. However, this effect is not too pronounced with the augmented tf, because the highest frequency term cannot degrade the frequency of other terms below 0.5. The logarithmic term frequency reduces the effect of unusually frequent terms within a document, and also the importance of raw term frequency in a collection of documents with significant variations in length. It actually decreases the effect of all sorts of variations in tf, because for any two term frequencies tf_1 and $tf_2 > 0$, such that $tf_1 > tf_2$, the ratio of the logarithmic term frequencies will be always less than the ratio of the raw term frequencies, i.e.,

$$\frac{\log(tf_1)+1}{\log(tf_2)+1} < \frac{tf_1}{tf_2}$$

Different choices for *A*, *B*, and *C* for query and document vectors yield different retrieval modes, for example, ntc-ntc, lnc-ltc, etc. The choices for tf are *n* (use the raw term frequency), *b* (binary, i.e., neglect term frequency, term frequency will be 1 if term is present in the document, otherwise 0), *a* (augmented normalized frequency), *l* (logarithmic term frequency), and *L* (logarithmic frequency normalized by average term frequency). The options for idf are *n* (use 1.0, ignore idf factor) and *t* (use idf). The possible options listed in Table 9.3 for document length normalization are *n* (no normalization) and *c* (cosine normalization). To achieve cosine normalization, every element of the term weight vector is divided by the Euclidean length of the vector. This is called cosine normalization because the length of the normalized vector is 1 and its projection on any axis in document space gives the cosine of the angle between the vector and the axis under consideration.

The widely known weighting scheme, ntc-ntc, normalizes both the document and query term weight in the range 0–1 and may prove

beneficial. The weighting scheme lnc-ltc means that document term weights are computed as the product of the logarithmic tf (*l*) of the given term, 1.0 (*n*) and cosine normalization (*c*) of the document vector. The query term weights are computed in the same way, except that each query term weight is also multiplied by the idf (*t*) of the given term in the document collection.

More recent weighting schemes integrate document length within the weighting formula yielding more complex retrieval models, for example, Okapi probabilistic search model (Robertson et al. 1995) and doc= "Lnu" model (Buckley et al. 1996). In TREC-3, the three systems with the best base performance were Okapi, INQUERY, and Cornell's Smart. The best performance was reported by Okapi system. Okapi uses the BM25 weighting algorithm introduced by developers of the probabilistic model during TREC-2 (Robertson et al. 1994) and TREC-3 (Robertson et al. 1995). Robertson and Walker (1994) developed the best match (BM) algorithms using the probabilistic model and some simple approximations to two-poisson model.

Considerable research efforts have been devoted to refining term weighting methods. As a result, a large number of term weighting schemes have been proposed in IR literature. (Salton and McGill 1983, Rijsbergen 1979). Some recent weighting schemes also consider the structure of the document.

A simple automatic method for obtaining indexed representation of the documents is as follows.

Step 1 Tokenization This extracts individual terms form a document, converts all the letters to lower case, and removes punctuation marks. The output of the first stage is a representation of the document as a stream of terms.

Step 2 Stop word elimination This removes words that appear more frequently in the document collection.

Step 3 Stemming This reduces the remaining terms to their linguistic root, to obtain the index terms.

Step 4 Term weighting This assigns weights to terms according to their importance in the document, in the collection, or some combination of both.

Table 9.4 shows the document vectors obtained after the application of these steps on sample documents shown in Figure 9.3.

Document 1:	Vector space model
Document 2:	Probabilistic retrieval model
Document 3:	Intelligent techniques in information retrieval

Figure 9.3 Sample documents

Table 9.4 Vector representation of sample documents after stemming

Stemmed terms	Document 1	Document 2	Document 3
inform	0	0	1
intellig	0	0	1
model	1	1	0
probabilist	0	1	0
retriev	0	1	1
space	1	0	0
technique	0	0	1
vector	1	0	0

9.4.5 Similarity Measures

Vector space model represents documents and queries as vectors in a multi-dimensional space. Retrieval is performed by measuring the 'closeness' of the query vector to document vector. Documents can then be ranked according to the numeric similarity between the query and the document. In the vector space model, the documents selected are those that are geometrically closest to the query according to some measure. The model relies on the intuitive notion that similar vectors define semantically related documents. Figure 9.4 gives an example of document and query representation in two-dimensional vector space. These dimensions correspond to the two index terms t_i and t_j. Document d_1 has two occurrences of t_i, document d_2 has one occurrence of t_i, and document d_3 has one occurrence of t_i and t_j each. Documents d_1, d_2, and d_3 are represented in this space using term weights—raw term frequency being used here—as coordinates. The angles between the documents and query are represented as θ_1, θ_2, and θ_3 respectively.

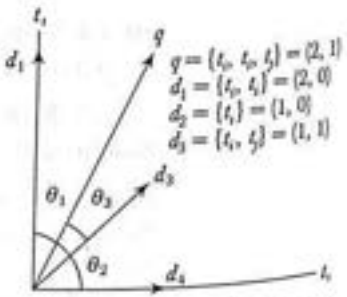


Figure 9.4 Representation in two-dimensional vector space

The simplest way of comparing document and query is by counting the number of terms they have in common. One frequently used similarity measure

is to take the 'inner product' between the query and the document vector. The inner product is given by

$$\text{sim}\,(d_j, q_k) = (d_j, q_k) = \sum_{i=1}^{m} w_{ij} \times w_{ik}$$

where m is the number of terms used to represent documents in the collection.

Other measures, e.g., dice coefficient, Jaccard's coefficient, and cosine coefficient, attempt to normalize the similarity by the length of document and query. The dice coefficient defines the similarity between query k and document j as

$$\text{sim}\,(d_j, q_k) = \frac{2 \times \left(\sum_{i=1}^{m} w_{ij} \times w_{ik}\right)}{\sum_{i=1}^{m} w_{ij}^2 + \sum_{i=1}^{m} w_{ik}^2}$$

Jaccard's coefficient is defined as

$$\text{sim}\,(d_j, q_k) = \frac{\sum_{i=1}^{m} w_{ij} \times w_{ik}}{\sum_{i=1}^{m} w_{ij}^2 + \sum_{i=1}^{m} w_{ik}^2 - \sum_{i=1}^{m} w_{ij} \times w_{ik}}$$

The cosine measure is commonly used for measuring similarity in IR. It computes cosine of the angle between the document and query vector, to give a similarity value between 0 and 1. A minimum value of 0 (angle 90°) indicates that the vectors are unrelated (i.e., they have no terms in common), and a value of 1 means that the vectors share common terms. If d_j and q_k are the document and query vector respectively, then the cosine similarity is computed as

$$\text{sim}\,(d_j, q_k) = \frac{(d_j, q_k)}{\|d_j\|\|q_k\|} = \frac{\sum_{i=1}^{m} w_{ij} \times w_{ik}}{\sqrt{\sum_{i=1}^{m} w_{ik}^2} \times \sqrt{\sum_{i=1}^{m} w_{ij}^2}}$$

If both the document and the query vectors have been cosine-normalized, then the inner product yields the cosine similarity.

The cosine measure divides the numerator by the product of the length of vectors. This tends to give low similarities to long vectors, i.e. vectors with many terms. The overlap coefficient compensates for this by dividing

by the vector having smaller sum of weights. More formally, this measure is defined as

$$\text{sim}(d_j, q_k) = \frac{\sum_{i=1}^{m} w_{ij} \times w_{ik}}{\min\left(\sum_{i=1}^{m} w_{ij}, \sum_{i=1}^{m} w_{ik}\right)}$$

9.5 NON-CLASSICAL MODELS OF IR

Non-classical IR models are based on principles other than similarity, probability, Boolean operations, etc., on which classical retrieval models are based. Examples include information logic model, situation theory model, and interaction model.

The *information logic model* is based on a special logic technique called logical imaging. Retrieval is performed by making inferences from document to query. This is unlike classical models, where a search process is used. Unlike usual implication, which is true in all cases except that when antecedent is true and consequent is false, this inference is uncertain. Hence, a measure of uncertainty is associated with this inference. The principle put forward by van Rijsbergen is used to measure this uncertainty. This principle says:

Given any two sentences x and y, a measure of the uncertainty of $y \rightarrow x$ relative to a given data set is determined by the minimal extent to which one has to add information to the data set in order to establish the truth of $y \rightarrow x$.

In fact, this model was developed in response to van Rijsbergen's realization that classical models were unable to enhance effectiveness and that new meaning-based models were required to do so.

The *situation theory* model is also based on van Rijsbergen's principle. Retrieval is considered as a flow of information from document to query. A structure called *infon*, denoted by ι, is used to describe the situation and to model information flow. An infon represents an n-ary relation and its polarity. The polarity of an infon can be either 1 or 0, indicating that the infon carries either positive or negative information.

For example, the information in the sentence, *Adil is serving a dish*, is conveyed by the infon

ι = ⟨⟨serving Adil, dish; 1⟩⟩

The polarity of an infon depends on the support. The support of an infon is represented as $s \models \iota$ and means that the situation s makes the infon ι true. For example, the infon, ι = ⟨⟨*serving Adil, dish*; 1⟩⟩ is made true by a situation $s1$ = "I see Adil serving a dish."

A document d is relevant to a query q, if $d \models q$

If a document does not support the query q, it does not necessarily mean that the document is not relevant to the query. Additional information, such as synonymys, hypernyms/hyponyms, meronyms, etc., can be used to transform the document d into d' such that $d' \models q$. Semantic relationships in a thesaurus, like WordNet, are useful sources for this information. The transformation from d to d' is regarded as flow of information between situations.

The *interaction IR* model was first introduced in Dominich (1992, 1993) and Rijsbergen (1996). In this model, the documents are not isolated; instead, they are interconnected. The query interacts with the interconnected documents. Retrieval is conceived as a result of this interaction. This view of interaction is taken from the concept of interaction as realized in the Copenhagen interpretation of quantum mechanics. Artificial neural networks can be used to implement this model. Each document is modelled as a neuron, the document set as a whole forms a neural network. The query is also modelled as a neuron and integrated into the network. Because of this integration, new connections are built between the query and the documents, and existing connections are changed. This restructuring corresponds to the concept of interaction. A measure of this interaction is obtained and used for retrieval. Detailed mathematical treatments of the model have been discussed by Dominich (1992, 1993, 2001) and van Rijsbergen (1996).

9.6 ALTERNATIVE MODELS OF IR

9.6.1 Cluster Model

The cluster model is an attempt to reduce the number of matches during retrieval. The need for clustering was first pointed out by Salton. Before we discuss the cluster-based IR model, we would like to state the cluster hypothesis that explains why clustering could prove efficient in IR.

Closely associated documents tend to be relevant to the same clusters.

This hypothesis suggests that closely associated documents are likely to be retrieved together. This means that by forming groups (classes or clusters) of related documents, the search time reduced considerably. Instead of matching the query with every document in the collection, it is matched with representatives of the class, and only documents from a class whose representative is close to query, are considered for individual match.

Clustering can be applied on terms instead of documents. Thus, terms can be grouped to form classes of co-occurrence terms. Co-occurrence terms can be used in dimensionality reduction or thesaurus construction. A number of methods are used to group documents. We discuss here, a cluster generation method based on similarity matrix. This method works as follows:

Let $D = \{d_1, d_2, \ldots, d_j, \ldots, d_m\}$ be a finite set of documents, and let $E = (e_{ij})_{n,n}$ be the similarity matrix. The element E_{ij} in this matrix, denotes a similarity between document d_i and d_j. Let T be the threshold value. Any pair of documents d_i and d_j $(i \neq j)$ whose similarity measure exceeds the threshold $(e_{ij} \geq T)$ is grouped to form a cluster. The remaining documents form a single cluster. The set of clusters thus obtained is

$$C = \{C_1, C_2, \ldots, C_k, \ldots, C_p\}$$

A representative vector of each class (cluster) is constructed by computing the centroid of the document vectors belonging to that class. Representation vector for a cluster C_k is

$$r_k = \{a_{1k}, a_{2k}, \ldots, a_{ik}, \ldots, a_{mk}\}$$

An element a_{ik} in this vector is computed as

$$a_{ik} = \frac{\sum_{d_j \in C_k} a_{ij}}{|C_k|}$$

where a_{ij} is weight of the term t_i, of the document d_j, in cluster C_k. During retrieval, the query is compared with the cluster vectors

$$(r_1, r_2, \ldots, r_k, \ldots, r_p)$$

This comparison is carried out by computing the similarity between the query vector q and the representative vector r_k as

$$s_{ik} = \sum_{i=1}^{m} a_{ik} q_i, \quad k = 1, 2, \ldots, p$$

A cluster C_k whose similarity s_k exceeds a threshold is returned and the search proceeds in that cluster.

Example 9.4

Let

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

be the term-by-document matrix. The similarity matrix corresponding to these documents is

$$\begin{matrix} 1.0 & & \\ 0.9 & 1.0 & \\ 0.4 & 0.4 & 1.0 \end{matrix}$$

Using a threshold of 0.7, we get the following two clusters:

$$C_1 = \{d_1, d_2\}$$
$$C_2 = \{d_3\}$$

The cluster vectors (representatives) for C_1 and C_2 are

$$r_1 = (1\ 0.5\ 1\ 0\ 1)$$
$$r_2 = (0\ 0\ \ 1\ 1\ 0)$$

Retrieval is performed by matching the query vector with r_1 and r_2.

9.6.2 Fuzzy Model

In the fuzzy model, the document is represented as a fuzzy set of terms, i.e., a set of pairs $[t_i, \mu(t_i)]$, where μ is the membership function. The membership function assigns to each term of the document a numeric membership degree. The membership degree expresses the significance of term to the information contained in the document. Usually, the significance values (weights) are assigned based on the number of occurrences of the term in the document and in the entire document collection, as discussed earlier. Each document in the collection

$$D = \{d_1, d_2, \ldots, d_j, \ldots, d_n\}$$

can thus be represented as a vector of term weights, as in the following vector space model

$$(w_{1j}, w_{2j}, w_{3j}, \ldots, w_{ij}, \ldots, w_{mj})^t$$

where w_{ij} is the degree to which term t_i belongs to document d_j.

Each term in the document is considered a representative of a subject area and w_{ij} is the membership function of document d_j to the subject area represented by term t_i. Each term t_i is itself represented by a fuzzy set f_i in the domain of documents given by

$$f_i = \{(d_j, w_{ij})\} \mid i = 1, \ldots, m; j = 1, \ldots, n$$

This weighted representation makes it possible to rank the retrieved documents in decreasing order of their relevance to the user's query.

Typically, queries are Boolean queries. For each term that appears in the query, a set of documents is retrieved. Fuzzy set operators are then applied to obtain the desired result.

For a single-term query $q = t_q$, those documents from the fuzzy set $f_q = \{(d_j, w_{iq})\}$, are retrieved for which w_{iq} exceeds a given threshold. The threshold may also be zero.

Consider the case of an AND query $q = t_{q1} \wedge t_{q2}$

First, the fuzzy sets f_{q1} and f_{q2} are obtained and then, their intersection is obtained, using the fuzzy intersection operator $f_{q1} \vee f_{q2} = \min \{(d_j, w_{q1}), (d_j, w_{iq})\}$

The documents in this set are returned.

Similarly, for an OR query $q = t_{q1} \wedge t_{q2}$, the union of fuzzy sets f_{q1} and f_{q2} is computed to retrieve documents as follows:

$$f_{q1} \vee f_{q2} = \max \{(d_j, w_{iq1}), (d_j, w_{iq})\}$$

Example 9.5 Consider the following three documents:

d_1 = {information, retrieval, query}
d_2 = {retrieval, query, model}
d_3 = {information, retrieval}

where the set of terms used to represent documents is

T = {information, model, query, retrieval}

The fuzzy sets induced by these terms are

$f_1 = \{(d_1, 1/3), (d_2, 0) (d_3, 1/2)\}$
$f_2 = \{(d_1, 0), (d_2, 1/3) (d_3, 0)\}$
$f_3 = \{(d_1, 1/3), (d_2, 1/3) (d_3, 0)\}$
$f_4 = \{(d_1, 1/3), (d_2, 1/3) (d_3, 1/2)\}$

If the query is $q = t_2 \wedge t_4$, then document d_2 will be returned.

9.6.3 Latent Semantic Indexing Model

Latent semantic indexing model is the application of single value decomposition to IR. The use of latent semantic indexing (LSI) is based on the assumption that there is some underlying 'hidden' semantic structure in the pattern of word-usage across documents, rather than just surface level word choice. LSI attempts to identify this hidden semantic structure through statistical techniques and use it to represent and retrieve information. This is done by modelling the association between terms and documents based on the manner in which terms co-occur across documents. LSI transforms the term-document vector space into a more compact latent semantic space. Each dimension in the reduced space corresponds to an 'artificial concept'. These concepts loosely correspond to a set of terms. It is believed that in the vector space of reduced dimensionality, the words referring to related concepts, i.e., words that

co-occur, are collapsed into the same dimension. Latent semantic space is thus able to capture similarities that go beyond term similarity. In the latent semantic space, a query and a document can have high similarity even if the document does not contain a query term, provided the terms are semantically related.

Now we discuss how the LSI technique is actually employed in IR. The document collection is first processed to get a $m \times n$ term-by-document matrix, W, where m is the number of index terms and n is the total number of documents in the collection. Columns in this matrix represent document vectors, whereas the rows denote term vectors. The matrix element W_{ij} represents the weight of the term i in document j. The weight may be assigned based on term frequency or some combination of local and global weighting, as in the case of vector space model. Singular value decomposition (SVD) of the term-by-document matrix is then computed. Using SVD, the matrix is represented as a product of three matrices

$$W = TSD^T$$

where T corresponds to term vectors and has m rows and r columns and $r = \min (m, n)$. S corresponds to singular values. D^T is the transpose of D and has r rows and n columns. D corresponds to the document vector.

T and D are orthogonal matrices containing the left and right singular vectors of W. S is a diagonal matrix, containing singular values stored in decreasing order. We eliminate small singular values and approximate the original term-by-document matrix using truncated SVD. For example, by considering only the first k number of the largest singular values, along with their corresponding columns in T and D, we get the following approximation of the original term-by-document matrix in a space of k orthogonal dimensions, where k is sufficiently less than n:

$$W_k = T_k S_k D_k^T$$

where T_k is the first k columns of T, D_k^T is the first k columns of D^T, and S_k is the k largest singular values.

The matrix W_k is used for retrieval. The idea is that the elimination of small singular values throws out the 'noise' resulting from term usage variation, and captures the underlying 'hidden' semantic structure (i.e., concepts). Each dimension in the reduced space corresponds to artificial or derived concepts. Each such concept loosely represents a set of terms in the original term-document matrix. Documents with varying word usage patterns are collapsed to the same vector in k-space.

The queries are also represented in k-dimensional space. Let $q = (q_1, q_2, \ldots, q_m)$ be the original query vector, where each element q_i is the frequency

of term i in the query q. The query q is represented in the k-dimensional space as

$$q_k = q^T T_k S_k^{-1}$$

where q^T is the transpose of the query vector, and T_k and S_k are the weights. $q^T T_k$ denotes the sum of k-dimensional term vectors and S_k^{-1}, the weights of each dimension. Thus, the query is represented as the weighted sum of its constituent term vectors.

Retrieval is performed by computing the similarity between query vector and document vector. For example, we can use the cosine similarity measure to rank documents to perform retrieval. In a keyword-based retrieval, relevant documents that do not share any term with the query are not retrieved. The LSI-based approach is capable of retrieving such documents, as similarity is computed based on the overall pattern of term usage across the document collection rather than on term overlap.

We now give an example to explain how a document in high-dimensional space is represented in a low, reduced, latent semantic space.

Example 9.6 Consider the matrix shown in Figure 9.5. This matrix defines five-dimensional space in which six documents, $d_1, d_2, d_3, \ldots, d_6$, have been represented. The five dimensions correspond to five index terms *tornado, storm, tree, forest,* and *farming*. For simplicity, tf has been used to weight index terms. Figure 9.6 shows the documents in a two-dimensional space. The vectors in the figure correspond to document vectors in the matrix R, which is the representation of X in reduced two-dimensional space. The two dimensions correspond to derived concepts obtained through the application of truncated SVD.

$$X = \begin{array}{c} \\ \text{tornado} \\ \text{storm} \\ \text{tree} \\ \text{forest} \\ \text{farming} \end{array} \begin{array}{c} \begin{array}{cccccc} d_1 & d_2 & d_3 & d_4 & d_5 & d_6 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

Figure 9.5 A term-document matrix representing six documents in five-dimensional space

We now explain how to arrive at the reduced dimensionality representation of X. First, the SVD of X is computed to get the three matrices T, S, and D.

$$X_{5\times 6} = T_{5\times 5}\, S_{5\times 5}\, (D_{6\times 5})^T$$

These matrices are shown in Figures 9.7, 9.8, and 9.9 respectively. Consider the first two largest singular values of S, and rescale $D^T_{2\times 6}$ with singular

values to get matrix $R_{2\times 6} = S_{2\times 2} D^T_{2\times 6}$, as shown in Figure 9.10, where $S_{2\times 2}$ is S restricted to two dimensions and $D^T_{2\times 6}$ is D^T restricted to two columns. R is a reduced dimensionality representation of the original term-by-document matrix X and is used to plot the vectors in Figure 9.6.

To find out the changes introduced by the reduction, we compute document similarities in the new space and compare them with the similarities between documents in the original space. The document-document correlation matrix for the original n-dimensional space is given by the matrix $Y = X^T X$. Here, Y is a square, symmetric $n \times n$ matrix. An element Y_{ij} in this matrix gives the similarity between documents i and j. The correlation matrix for the original document vectors is shown in Figure 9.12. This matrix is computed using X, after normalizing the lengths of its columns. The document-document correlation matrix for the new space is computed analogously using the reduced representation R. Let N be the matrix R with length-normalized columns. Then, $M = N^T N$ gives the matrix of document correlations in the reduced space. The correlation matrix M is given in Figure 9.11. The similarity between document d_1, d_4(−0.0304), and d_6(−0.2322) is quite low in the new space because document d_1 is not topically similar to documents d_4 and d_6. In the original space, the similarity between documents d_2 and d_3 and between documents d_2 and d_5 is 0. In the new space, they have high similarity values (0.5557 and 0.8518 respectively) although documents d_3 and d_5 share no term with the document d_2. This topical similarity is recognized due to the co-occurrence of patterns in the documents.

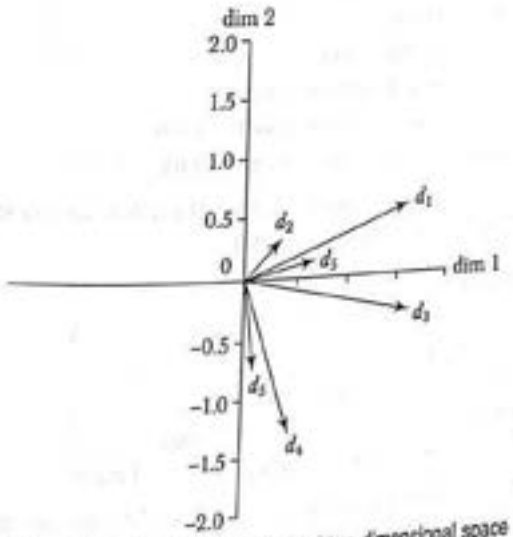


Figure 9.6 Documents in reduced two-dimensional space

$$T = \begin{pmatrix} 0.3318 & 0.3338 & 0.8064 & -0.2426 & -0.2634 \\ 0.6693 & 0.1616 & -0.2737 & 0.5853 & -0.3293 \\ 0.5514 & 0.1038 & -0.0961 & -0.2667 & 0.7777 \\ 0.3583 & -0.5745 & -0.2148 & -0.5778 & -0.4021 \\ 0.0974 & -0.7223 & 0.4684 & 0.4400 & 0.2362 \end{pmatrix}$$

Figure 9.7 Matrix T for the SVD of the term-document matrix X shown in Figure 9.5

$$S = \begin{pmatrix} 2.3830 & 0 & 0 & 0 & 0 \\ 0 & 1.6719 & 0 & 0 & 0 \\ 0 & 0 & 1.2415 & 0 & 0 \\ 0 & 0 & 0 & 0.8288 & 0 \\ 0 & 0 & 0 & 0 & 0.5454 \end{pmatrix}$$

Figure 9.8 The matrix S for singular values of the SVD of the term-document matrix X

$$D^T = \begin{pmatrix} 0.6515 & 0.1392 & 0.6626 & 0.1912 & 0.2809 & 0.0409 \\ 0.3584 & 0.1996 & -0.1848 & -0.7756 & 0.0967 & -0.4320 \\ 0.3516 & 0.6895 & -0.4710 & 0.2042 & -0.2205 & 0.3773 \\ 0.0916 & -0.2927 & -0.3127 & -0.1662 & 0.7062 & 0.5309 \\ 0.3392 & -0.4829 & 0.0849 & -0.3042 & -0.6037 & 0.4330 \end{pmatrix}$$

Figure 9.9 The matrix D^T for singular values of the SVD of the term-document matrix

$$R = \begin{pmatrix} 1.5526 & 0.3318 & 1.5790 & 0.4557 & 0.6693 & 0.0974 \\ 0.5992 & 0.3338 & -0.3090 & -1.2967 & 0.1616 & -0.7223 \end{pmatrix}$$

Figure 9.10 The matrix $R_{2\times6} = S_{2\times2}D^T_{2\times6}$ representing documents in two-dimensional space

$$M = \begin{pmatrix} 1.0000 & & & & & \\ 0.9131 & 1.0000 & & & & \\ 0.8464 & 0.5557 & 1.0000 & & & \\ -0.0304 & -0.4353 & 0.5066 & 1.0000 & & \\ 0.9914 & 0.8518 & 0.9089 & 0.1008 & 1.0000 & \\ -0.2322 & -0.6085 & 0.3215 & 0.9793 & -0.1027 & 1.0000 \end{pmatrix}$$

Figure 9.11 The matrix of document correlation $M = N^TN$ in the new space (N is matrix R with length-normalized columns.)

$$Z = \begin{pmatrix} 1.0000 & & & & & \\ 0.5774 & 1.0000 & & & & \\ 0.6667 & 0 & 1.0000 & & & \\ 0 & 0 & 0.4082 & 1.0000 & & \\ 0.5774 & 0 & 0.5774 & 0 & 1.0000 & \\ 0 & 0 & 0 & 0.7071 & 0 & 1.0000 \end{pmatrix}$$

Figure 9.12 The matrix of document correlation $Z = Y^TY$ in the new space (Y is matrix X with length-normalized columns.)

The LSI performs IR based on concept. It is completely automatic and has been applied successfully (Deerwester et al. 1990, Foltz 1990) in many IR systems. However, it is costly in terms of computation.

9.7 EVALUATION OF THE IR SYSTEM

The evaluation of IR systems is the process of assessing how well a system meets the information needs of its users (Voorhees 2001). Evaluating an IR system is a difficult task involving a number of areas including cognition, statistics, and man-machine interactions. IR evaluation models can be broadly classified as system driven models and user-centered models. System driven models (Cleverdon et al. 1966) measure how well a system ranks documents; user-centered models measure user satisfaction. Cleverdon listed the following six criteria that can be used for evaluation:

1. *Coverage of the collection*: The extent to which the system
2. *Time lag*: The time that elapses between submission of a query and getting back the response
3. *Presentation format*
4. *User effort*: The effort made by the user to obtain relevant information
5. *Precision*: The proportion of retrieved documents that are relevant
6. *Recall*: The proportion of relevant documents that are retrieved

Of these criteria, recall and precision have most frequently been applied in measuring IR. Both are related to effectiveness, i.e., the ability of a system to retrieve relevant documents in response to user query. A number of effectiveness measures have been formulated (van Rijsbergen 1979). We discuss them in the following section. To better understand the relationship between aspects of retrieval process and different measures, see (Voorhees and Harman 1999), where correlations between pairs of measures are estimated.

The major goal of IR is to search for documents that are relevant to a user's query. It is necessary to understand what constitutes relevance, as the evaluation of IR systems relies on the notion of relevance.

9.7.1 Relevance

Relevance is subjective in nature (Saracevic 1991), i.e., it depends on the individual judgements of users. Given a query, the same document may be judged as relevant by one user and non-relevant by another. It is not possible to measure this 'true relevance' because no human can read all documents in a collection and provide a relevance assessment. Most evaluations of IR systems have so far been done on test document

collections with known relevance judgments. These test document collections contain documents from a particular discipline; for a set of questions representing information needs, relevance assessments are obtained from experts of that discipline. This provides an experimental setup for evaluating the performance of a retrieval strategy. If a retrieval strategy performs well under these situations, it is expected to perform well in an operational environment where relevance is not known.

Another issue with relevance is the degree of relevance. Traditionally, relevance has been visualized as a binary concept, i.e., a document is either relevant or not relevant; whereas relevance is actually a continuous function (a document may be exactly what the user wants or it may be closely related). This is an attractive but difficult proposition and current evaluation techniques do not support it.

A number of relevance frameworks have been proposed by Saracevic (1996). This includes system, communication, psychological, and situational frameworks. The most inclusive of these is the situational framework, which is based on a cognitive view of the information seeking process and considers the importance of situation, context, multi-dimensionality, and time. A survey of relevance studies has been discussed by Mizzarro, (1996).

9.7.2 Effectiveness Measures

Effectiveness is purely a measure of the ability of a system to satisfy the user in terms of the relevance of documents retrieved (Rijsbergen 1979). Aspects of effectiveness include whether the documents returned are relevant to the user, whether they are presented in order of relevance, whether a significant number of relevant documents in the collection are returned to the user, etc. A number of measures have been proposed to quantify effectiveness. As stated earlier, the most commonly used measures of effectiveness are precision and recall. These measures are based on relevance judgments.

Precision and Recall

Precision is defined as the proportion of relevant documents in a retrieved set. This can be seen as the probability that a relevant document is retrieved. *Recall* is the proportion of relevant documents in a collection that have actually been retrieved. Precision measures the accuracy of a system while recall measures its exhaustiveness. Precision and recall can be computed as follows:

$$\text{Precision} = \frac{\text{Number of relevant document retrieved } (NR_{ret})}{\text{Total number of documents retrieved } (N_{ret})}$$

$$\text{Recall} = \frac{\text{Number of relevant documents retrieved } (NR_r)}{\text{Total number of relevant documents in the collection } (NR_{rel})}$$

These definitions of precision and recall are based on binary relevance judgment, which means that every retrievable item is recognizably 'relevant', or recognizably 'not relevant'. Hence, for every search result, all retrievable documents will be either (i) relevant or non-relevant and (ii) retrieved or not retrieved. Thus, each document will fall into one, and only one, of four cells of the matrix, as sown in Figure 9.13. This matrix is used to derive a number of measures.

	Relevant	Non-relevant	
Retrieved	$A \cap B$	$\bar{A} \cap B$	B
Not-retrieved	$A \cap \bar{B}$	$\bar{A} \cap \bar{B}$	$\bar{B}$
	A	$\bar{A}$	

Figure 9.13 Relevant matrix

Referring to Figure 9.13, precision and recall will be given as follows:

$$\text{Precision} = \frac{|A \cap B|}{|B|} = \frac{NR_{ret}}{N_{ret}}$$

$$\text{Recall} = \frac{|A \cap B|}{|A|} = \frac{NR_{ret}}{NR_{rel}}$$

where A = Set of relevant documents

$|A|$ = No. of relevant documents in the collection (NR_{rel})

B = Set of retrieved documents

$|B|$ = No. of retrieved documents (NR_{ret})

It is clear from the preceding definitions that the total number of relevant documents in a collection must be known in order for recall to be calculated. The amount of effort and time required from the user makes this almost impossible in most operating environment. To provide a framework of evaluation for IR systems, a number of test collections have been developed (Cranfield and TREC). These collections are accompanied by a set of queries and relevance judgements. These test

collections make it possible for IR researchers to efficiently evaluate their experimental approaches and compare the effectiveness of their system with that of others. In Table 9.5, basic statistics for a number of test collections are presented.

Table 9.5 IR test collections

Collection	Number of documents	Number of queries
Cranfield	1400	225
CACM	3204	64
CISI	1460	112
LISA	6004	35
TIME	423	83
ADI	82	35
MEDLINE	1033	30
TREC-1	742,611	100

There exists a trade-off between precision and recall, though a high value of both at the same time is desirable. The trade-off is shown in Figure 9.14. Precision is high at low recall values. As recall increases, precision decreases. The ideal case of perfect retrieval requires that all relevant documents be retrieved before the first non-relevant document is retrieved. This is shown in the figure by the line parallel to *x*-axis having a precision of 1.0 at all recall points. Recall is an additive process. Once the highest recall (1.0) is achieved, it remains 1.0 for any subsequent document retrieved. We can always achieve 100% recall by retrieving all documents in the collection, but this defeats the intent of an IR system.

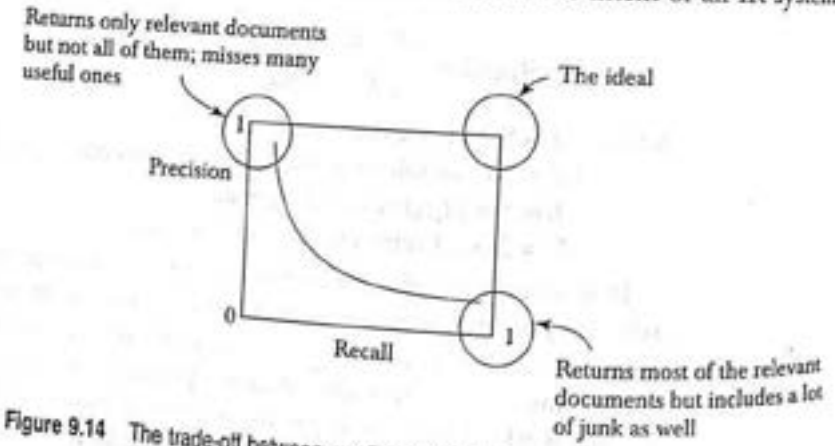


Figure 9.14 The trade-off between recall and precision

A number of researchers have discussed the relationship between recall and precision (Cleverdon 1972, Robertson 1975, Gordon and Kochen 1989, Buckland and Gey 1994). Some of them modelled precision and recall as continuous functions (Robertson 1975, Gordon and Kochen 1989), while others (Bookstein 1974) described recall and precision in terms of two-Poisson discrete model. Buckland and Gey (1994) studied the relationship between precision and recall, and suggested that a two-stage, or more generally, a multi-stage retrieval procedure is likely to achieve the goal of improving both precision and recall simultaneously, even though the trade-off between them cannot be avoided.

In order to evaluate the performance of an IR system, recall and precision are almost always used together. One measure is to calculate precision at a particular cut-off. Typical cut-offs are 5 documents, 10 documents, 15 documents, etc.

Yet another measure is *non-interpolated average precision*, which is average of the precision at observed recall points. Observed recall points correspond to points where a relevant document is retrieved. We first compute precision at each point where a relevant document is found and then compute average of these precision numbers to get a single number. Precision at relevant documents not in the returned set is assumed to be zero. We now give an example to illustrate how precision is calculated.

Table 9.6 An example of retrieval

Rank	Document #	Relevance
1	10	x
2	8	x
3	5	
4	3	
5	1	x
6	2	
7	4	
8	7	
9	9	x
10	6	x

Example 9.7 Table 9.6 shows the ranking of 10 documents for a particular retrieval. The crossed documents are those that are relevant. Let the total number of relevant document be five.

Precision values at 5 and 10 documents are given as follows:

Precision at 5 $3/10 = 0.3$

Precision at 10 $5/10 = 0.5$

Non-interpolated Average Precision The observed recall points are 0.2, 0.4, 0.6, 0.8, and 1.0. These recall values correspond to the documents marked relevant in the table. We have one of five relevant documents retrieved after retrieving only one document. This correspond to a recall value of 1/5 = 0.2. After two documents, the recall is 2/5 = 0.4. As the third document retrieved is not relevant, recall value does not change. The next relevant document is found after five documents have been retrieved, resulting in a recall value of 3/5 = 0.6. Similarly, the other two recall points are calculated. The precision values at these points are as follows:

Precision at recall point 0.2: 1/1 = 1.0
Precision at recall point 0.4: 2/2 = 1.0
Precision at recall point 0.6: 3/5 = 0.6
Precision at recall point 0.8: 4/9 = 0.4
Precision at recall point 1.0: 5/10 = 0.5
Non-interpolated average precision = 0.7

Considering the fact that a system may not always retrieve all the relevant documents, and that the number of relevant documents is not the same for all queries, precision values are interpolated for a set of recall points. The most widely used recall levels are 0.0, 0.1, 0.2, 0.3... 1.0. The precision values are calculated at each of these 11 recall levels and then averaged to get a single value. This is known as 11-point interpolated average precision. This has become almost a standard in evaluating the performance of an IR system. The interpolation used at TREC states that, precision at a given recall level is the greatest known precision at any recall level greater than or equal to this given level. For example, if the observed recall points are 0.25, 0.4, 0.55, 0.8, and 1.0, then precision at recall level 0.3 will be the maximum of the precision at recall levels 0.4, 0.55, 0.8, and 1.0, and not precision at recall point 0.4 where the 30% recall (i.e., recall level 0.3) is first reached. The interpolated precision at standard recall points for the documents shown in the Table 9.5 is computed in Example 9.8.

Example 9.8 Consider the following precision values at observed recall points:

0.25	1.0
0.4	0.67
0.55	0.8
0.8	0.6
1.0	0.5

The interpolated precision at the standard 11 recall levels will be

0.0	1.0
0.1	1.0
0.2	1.0
0.3	0.8
0.4	0.8
0.5	0.8
0.6	0.6
0.7	0.6
0.8	0.6
0.9	0.5
1.0	0.5

Interpolated average precision = 0.745

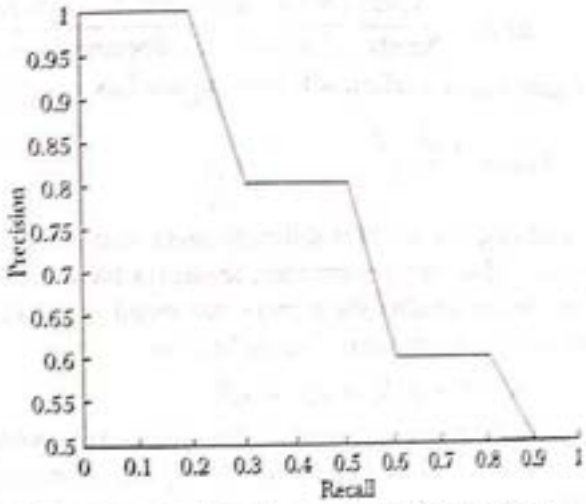


Figure 9.15 Recall–precision curve for interpolated precision

Often, precision values are calculated at different recall levels and a recall–precision graph, like the one shown in Figure 9.15, is plotted. As a retrieval system is evaluated over several queries, such a graph is usually plotted using precision figures averaged over all queries (Salton and McGill 1983, van Rijsbergen 1979). The most standard method for deriving a recall–precision graph is to plot the average over all queries in the interpolated precision values for a set of 11 standard recall points, namely 0.0, 0.1, 0.2, 0.3, ..., 1.0.

Instead of the recall–precision graph, the mean average precision is sometimes used to evaluate an IR system. The non-interpolated average

precision is averaged over all queries to get the mean average precision (MAP). Geometrically, MAP is the area below the non-interpolated recall-precision curve (Voorhees and Harman 1999). The 11-point interpolated average precisions (11 avgP) can also be used for calculating MAP, though the non-interpolated measure has the advantage that it rewards systems that quickly retrieve (give high ranks to) relevant documents.

The R-precision is the precision after a total number of R documents relevant to the query have been retrieved.

Recall is not defined if there is no relevant document in a collection. An alternative measure is fallout, which may be seen as the inverse of recall. It is not defined only if all the documents in the collection are relevant (Salton 1983). Fallout is the ratio of non-relevant documents retrieved to non-relevant documents in the collection.

$$\text{Fallout} = \frac{\text{Number of non-relevant documents retrieved } (N_n)}{\text{Number of non-relevant documents in the collection}}$$

For Figure 9.13, the fallout will be computed as

$$\text{Fallout} = \frac{|\bar{A} \cap B|}{|\bar{A}|}$$

Considering the fact that different users may have different ideas of relevance and system performance, measures have been developed to let the user decide whether she is more interested in recall or precision. For instance, the utility measure U is defined as

$$U = \alpha \cdot N_r + \beta \cdot \bar{N}_r + \delta N_n + \gamma \bar{N}_n$$

where N_r is the number of relevant documents retrieved, $\bar{N}_r$ the number of relevant documents not retrieved, N_n the number of non-relevant documents retrieved, and $\bar{N}_n$ the number of non-relevant documents not retrieved (α, β, δ, and γ are positive weights specified by the user). This measure was later simplified by considering only retrieved documents.

The F-measure takes into account both precision and recall. It is defined as the harmonic mean of recall and precision.

$$F = \frac{2PR}{P + R}$$

Compared to the arithmetic mean, both recall and precision need to be high for harmonic mean to be high.

The E-measure is a variant of the F-measure. It allows weighting to emphasize precision rather than recall. It is defined as

$$E = \frac{(1+\beta^2)PR}{\beta^2 P + R} = \frac{(1+\beta^2)}{\frac{\beta^2}{R} + \frac{1}{P}}$$

where P is precision, R is recall, and β is the relative importance of P compared to R. The value of β controls the trade-off between precision and recall. Setting β to 1 gives equal weight to precision and recall, resulting in a harmonic mean of recall and precision ($E = F$). $\beta > 1$ gives more weight to precision, and $\beta < 1$ gives more weight to recall.

Swets (1969) developed a model that received attention in the literature (Heine 1974; Bookstein 1977). None of these alternative measures, however, received as widespread acceptance as did the recall-precision model.

Normalized recall measures how close the set of retrieved documents is to an ideal retrieval, in which the most relevant NR_{rel} document appears in the first NR_{rel} position. Relevant documents are ranked 1, 2, 3, ..., NR_{rel}, where NR_{rel} is the number of relevant documents. The ideal rank is given by

$$\text{IdR} = \frac{\sum_{r=1}^{NR_{rel}} r}{NR_{rel}}$$

Let the average rank (AvR) over the set of relevant documents retrieved by a system be

$$\text{AvR} = \frac{\sum_{r=1}^{NR_{rel}} \text{Rank}_r}{NR_{rel}}$$

where Rank_r represents the rank of the rth relevant document. The difference between AvR and IdR given by AvR – IdR, represents a measure of the effectiveness of the system. This difference can range from 0, for the perfect retrieval (AvR – IdR), to ($N - NR_{rel}$), for the worst case (N is the total number of documents in the collection). The worst case is when all the N documents are retrieved and the relevant documents, NR_{rel}, are the last retrieved. The expression AR-IR can be normalized by dividing it by ($N - NR_{rel}$) and then subtracting the result from 1. The normalized recall (NR) is given by

$$\text{NR} = 1 - \frac{\text{AvR} - \text{IdR}}{N - NR_{rel}}$$

This measure ranges from 1 for the best case, to 0 for the worst case. If the value of NR is close to 1, the ranks of relevant documents in average case deviate very little from the ideal case. A high value of NR indicates that the ranks of the relevant documents in the average case deviate considerably from the ideal case.

9.7.3 User-centred Evaluation

The system-driven model is still the dominant approach followed in IR research for evaluation of IR systems. The evaluation here, is made on a test collection having known relevance judgments. These relevance judgements were usually provided by problem domain experts, and are binary, objective, topical, and static in nature, and lack a user's viewpoint. There is also major disagreement among experts in providing relevance judgements (Haynes et al. 1990, Hersh and Hickam 1994). Further, these judgements of relevance are affected not only by the expertise of the judge, but also by the order of the documents (Eisenberg and Barry 1988; Schamber et al. 1990). It has been argued that relevance is not fixed, that it varies over time (Meadow 1992). The meaning and the relevance of a document can thus be different for different users and can be inferred only in the context of the user's situation. Relevance, therefore, is subjective, dynamic, and multi-dimensional in nature (Saracevic 1975, Mizzaro 1998, Harter 1992).

Another drawback of the system-driven approach is that it removes the end users from the retrieval process, substituting them with the queries and judgements provided with the test collection. This allows fast experimentation but makes it difficult to evaluate the effect of interactive IR techniques and is suitable only for non-interactive environment (Draper and Dunlop). In an interactive setting, a user normally starts with a query, which goes through many refinements to eventually get the desired documents. The test-collection approach poses a problem in such an environment. As performance of the IR system will eventually be measured in terms of it ability to retrieve documents relevant to a user's query, it seems realistic to follow a user-centered approach to evaluation. Such an approach will result in a much more direct measure of the overall goal. A number of measures have been proposed for interactive IR including relative relevance (RR), ranked half life (RHL), and cumulated gain (CG). Details of these measures can be found in Hersh et al. (1995), Borlund and Ingwerson (1998), and Järvelin and Kekäläinen (2000).

The subjective nature of interactive IR has been highlighted (Borlund and Ingwerson 1997) and attempts have been made to integrate cognitive theory into IR evaluation. However, efforts in this direction have been limited. A task oriented, user-centred, non-interactive evaluation methodology has been proposed by Reid (2000), in which the basic unit of evaluation is task rather than query. More recently, an interactive IR evaluation model has been proposed by Borlund (2000, 2003) to evaluate interactive IR systems. The key elements of an interactive IR model are the use of realistic scenarios (simulated work tasks) and alternative performance measures such as RR and RHL. However, user-centred evaluation methods are expensive both in terms of time and resources. A properly designed user-centred evaluation, with a few exceptions, requires a sufficiently large, representative sample of actual users of retrieval systems. The systems to be compared must be equally well-developed and equipped with the appropriate user interface. The subject must be trained with these systems. Further, it is difficult to develop a standard interactive evaluation methodology that will allow for comparison across different systems and users (Reid 2000). Because of these considerations, recall and precision remain the most popular and standard measure for evaluating IR system performance.

SUMMARY

- Information retrieval (IR) deals with the organization, storage, retrieval, and evaluation of information relevant to a user's query.
- An IR system does not return the actual information but returns the documents containing that information.
- The actual text of the document is not used in the retrieval process. Instead, documents in a collection are frequently represented through a set of index terms or keywords.
- The process of transforming document text to some representation of it is known as *indexing*.
- A common lexical processing of index terms involves elimination of stop words.
- An IR model is a pattern that defines several aspects of the retrieval procedure, for example, how the documents and users' queries are represented, how the system retrieves relevant documents according to users' queries, and how retrieved documents are ranked.
- Classical IR models, such as Boolean, vector space, and probabilistic, are based on mathematical knowledge that is easily recognized and well understood.

CHAPTER 12

LEXICAL RESOURCES

CHAPTER OVERVIEW

This chapter introduces various tools and lexical resources used in text processing applications and provides a ready reference of these. In particular, it introduces the reader with tools such as stemmers and taggers, lexical resources such as WordNet and FrameNet, and test collections (corpuses) that are freely available for research purpose.

12.1 INTRODUCTION

A whole range of tools and lexical resources have been developed to ease the task of researchers working with natural language processing (NLP). Many of these are open sources, i.e., readers can download them off the Internet. This chapter introduces some of the freely available resources. The motivation behind including this chapter comes from the belief that *knowing where the information is, is half of the information.*

We hope that providing a ready reference of what is available where, will save a lot of time and effort, especially for young researchers and those who are new to the field. All the material presented in this chapter is already available, at the links provided with the discussion or in the form of scholarly articles published on that resource. We bring these resources together and offer a brief discussion on them. In particular, we discuss lexical resources such as WordNet and FrameNet, and tools such as stemmers, taggers, and parsers, and freely available test corpuses for various text-processing applications. We begin our discussion with WordNet in Section 12.2. Section 12.3 discusses FrameNet. Stemmers are discussed in Section 12.4. We present a list of available part-of-speech taggers in Section 12.5. The next section presents a list of document collections. And finally, relevant journals and conferences are listed in Section 12.7.

12.2 WORDNET

WordNet[1] (Miller 1990, 1995) is a large lexical database for the English language. Inspired by psycholinguistic theories, it was developed and is being maintained at the Cognitive Science Laboratory, Princeton University, under the direction of George A. Miller. WordNet consists of three databases—one for nouns, one for verbs, and one for both adjectives and adverbs. Information is organized into sets of synonymous words called *synsets*, each representing one base concept. The synsets are linked to each other by means of lexical and semantic relations. Lexical relations occur between word-forms (i.e., senses) and semantic relations between word meanings. These relations include synonymy, hypernymy/hyponymy, antonymy, meronymy/holonymy, troponymy, etc. A word may appear in more than one synset and in more than one part-of-speech. The meaning of a word is called sense. WordNet lists all senses of a word, each sense belonging to a different synset. WordNet's sense-entries consist of a set of synonyms and a gloss. A gloss consists of a dictionary-style definition and examples demonstrating the use of a synset in a sentence, as shown in Figure 12.1. The figure shows the entries for

12.1.1 Noun

1. **read** (something that is read) *"the article was a very good read"*

12.1.2 Verb

1. **read** (interpret something that is written or printed) *"read the advertisement"; "Have you read Salman Rushdie?"*
2. **read**, say (have or contain a certain wording or form) *"The passage reads as follows"; "What does the law say?"*
3. **read** (look at, interpret, and say out loud something that is written or printed) *"The King will read the proclamation at noon"*
4. **read**, scan (obtain data from magnetic tapes) *"This dictionary can be read by the computer"*
5. **read** (interpret the significance of, as of palms, tea leaves, intestines, the sky; also of human behaviour) *"She read the sky and predicted rain"; "I can't read his strange behavior"; "The fortune teller read his fate in the crystal ball"*
6. take, **read** (interpret something in a certain way; convey a particular meaning or impression) *"I read this address as a satire"; "How should I take this message?"; "You can't take credit for this!"*
7. learn, study, **read**, take (be a student of a certain subject) *"She is reading for the bar exam"*
8. **read**, register, show, record (indicate a certain reading; of gauges and instruments) *"The thermometer showed thirteen degrees below zero"; "The gauge read 'empty'"*
9. **read** (audition for a stage role by reading parts of a role) *"He is auditioning for 'Julius Caesar' at Stratford this year"*
10. **read** (to hear and understand) *"I read you loud and clear!"*
11. understand, **read**, interpret, translate (make sense of a language) *"She understands French"; "Can you read Greek?"*

Figure 12.1 WordNet 2.0 entry for 'read'

[1]http://wordnet.princeton.edu/

the word 'read'. 'Read' has one sense as a noun and 11 senses as a verb. Glosses help differentiate meanings. Figures 12.2, 12.3, and 12.4 show some of the relationships that hold between nouns, verbs, and adjectives and adverbs. Nouns and verbs are organized into hierarchies based on the hypernymy/hyponymy relation, whereas adjectives are organized into clusters based on antonym pairs (or triplets). Figure 12.5 shows a hypernym chain for 'river' extracted from WordNet. Figure 12.6 shows the troponym relations for the verb 'laugh'.

Relation	*Definition*	*Example*
Hypernym	From concepts to super-ordinates	oak → tree
Hyponym	From concepts to subtypes	oak → white oak
Meronym	From wholes to parts	tree → trunk
Holonym	From parts to wholes	trunk → tree
Antonym	Opposites	victory → defeat

Figure 12.2 Noun relations in WordNet

Relation	*Definition*	*Example*
Hypernym	From events to super-ordinate events	wander → travel
Troponym	From events to their subtypes	walk → stroll
Entails	From events to the events they entail	snore → sleep
Antonym	Opposites	increase → decrease

Figure 12.3 Verb relations in WordNet

Relation	*Definition*	*Example*
Antonym (adjective)	Opposite	heavy → light
Antonym (adverb)	Opposite	quickly → slowly

Figure 12.4 Adjective and adverb relations in WordNet

1 sense of 'river'
Sense 1
river — (a large natural stream of water (larger than a creek); 'the river was navigable for 50 miles')
=> stream, watercourse — (a natural body of running water flowing on or under the earth)
=> body of water, water — (the part of the earth's surface covered with water (such as a river or lake or ocean); 'they invaded our territorial waters'; 'they were sitting by the water's edge')
=> thing — (a separate and self-contained entity)
=> entity — (that which is perceived or known or inferred to have its own distinct existence (living or nonliving))

Figure 12.5 Hypernym chain for 'river'

WordNet is freely and publicly available for download from *http://wordnet.princeton.edu/obtain*.

WordNets for other languages have also been developed, e.g., EuroWordNet and Hindi WordNet. EuroWordNet covers European languages, including English, Dutch, Spanish, Italian, German, French, Czech, and Estonian. Other than language internal relations, it also contains multilingual relations from each WordNet to English meanings.

Hindi WordNet has been developed by CFILT (Resource Center for Indian Language Technology Solutions), IIT Bombay.[2] Its database consists of more than 26,208 synsets and 56,928 Hindi words.[3] It is organized using the same principles as English WordNet but includes some Hindi specific relations (e.g., causative relations). A total of 16 relations have been used in Hindi WordNet. Each entry consists of synset, gloss, and position of synset in ontology. Figure 12.7 shows the Hindi WordNet entry for the word 'आकांक्षा' (*aakanksha*).

Sense 1
laugh, express joy, express mirth — (produce laughter)
=> bray — (laugh loudly and harshly)
=> bellylaugh — (laugh a deep, hearty laugh)
=> roar, howl — (laugh unrestrainedly and heartily)
=> snicker, snigger — (laugh quietly)
=> giggle, titter — (laugh nervously; 'The girls giggled when the rock star came into the classroom')
=> break up, crack up — (laugh unrestrainedly)
=> cackle — (emit a loud, unpleasant kind of laughing)
=> guffaw, laugh loudly — (laugh boisterously)
=> chuckle, chortle, laugh softly — (laugh quietly or with restraint)
=> convulse — (be overcome with laughter)
=> cachinnate — (laugh loudly and in an unrestrained way)

Figure 12.6 Troponym relation for the word 'laugh'

Hindi WordNet can be obtained from the URL http://*www.cfilt.iitb.ac.in/wordnet/webhwn/*.

CFLIT has also developed a Marathi WordNet. Figure 12.8 shows the Marathi WordNet (*http://www.cfilt.iitb.ac.in/wordnet/webmwn/wn.php*) entry for the word 'पाव' (*pau*).

12.2.1 Applications of WordNet

WordNet has found numerous applications in problems related with IR and NLP. Some of these are discussed here.

[2]http://www.cfilt.iitb.ac.in/
[3]Hindi WordNet Documentation http://www.cfilt.iitb.ac.in/hindi-wordnet-license01.pdf

1) Concept Identification in Natural Language

WordNet can be used to identify concepts pertaining to a term, to suit them to the full semantic richness and complexity of a given information need.

2) Word Sense Disambiguation

WordNet combines features of a number of the other resources commonly used in disambiguation work. It offers sense definitions of words, identifies synstes of synonyms, defines a number of semantic relations and is freely available. This makes it the (currently) best known and most utilized resource for word sense disambiguation. One of the earliest attempts to use WordNet for word sense disambiguation was in IR by Voorheese (1993). She used WordNet noun hierarchy (hypernym / hyponym) to achieve disambiguation. A number of other researchers have also used WordNet for the same purpose (Resnik 1995, 1997; Sussna 1993).

1. (R) अपेक्षा, आकांक्षा, अन्वपेक्षा – किसी पर भरोसा रखने की क्रिया कि अमुक कार्य उसके द्वारा हो जायेगा " हर पिता की अपने पुत्र से यह अपेक्षा रहती है कि वह अपने जीवन में सफल हो "
2. (R) इच्छा, अभिलाषा, आकांक्षा, लालसा, आरजू, तमन्ना, वासना, तलब, चेष्टा, हसरत, मुराद, पिपासा, प्यास, तृष्णा, मनोकामना, मनोवासना, मनोरांछा, मनोरथ, मनोभावना, मरजी, रजा, मर्जी, मन, रजा, तंबा, लिप्सा, लालसा, तृष्ण, चाह, अरमान, क्षुधा, भूख, भूक्ष, क्षुधा, हवस, स्पृहा, अभीप्सा, अभु, अपेक्षित, अभिकांक्षा, अभिकाम, वांछा, वाञ्छा, वाञ्छा, अभिख्या, अभिलाष, अभिप्रीति, अभिमत, अभिमतता, अभिमति, अभिलाल, अभिलाता, अभिलाठा, अभिलाख, अभिलाखना — वह मनोवृत्ति जो किसी बात या वस्तु की प्राप्ति की ओर ध्यान से जाती है " इंसान की हर इच्छा पूरी नहीं होती / उसकी शान पिपासा बढ़ती जा रही है / मेरा आज खाने का मन नहीं है "

Figure 12.7 WordNet entry for the Hindi word आकांक्षा (aakanksha)

1. (R) पाव पुक चतुर्थांश-चौथा भाग "मी बाजारातून पुक पाव घुरहुरे आणले"
2. (R) पाव, पावरोटी-मैदाचे पीठ आंबवून केलेले पुक खाद्यविशेष "तुंबईत बरेच लोक पाव खाऊन गुजराण करतात"
3. (R) पाव-पावाचे वजन "दुकानदार चहाच्या पूडीचे वजन करण्यासाठी पाव मोजत आहे"
4. (R) पाव-पुक चतुर्थांश बाटली वाल "पुक पाव प्यालाभरून तो वळवट करू लागला"

Figure 12.8 WordNet entry for the Marathi word पाव (pau)

3) Automatic Query Expansion

WordNet semantic relations can be used to expand queries so that the search for a document is not confined to the pattern-matching of query terms, but also covers synonyms. The work performed by Voorhees (1994) is based on the use of WordNet relations, such as synonyms, hypernyms, and hyponyms, to expand queries.

4) ***Document Structuring and Categorization***

The semantic information extracted from WordNet, and WordNet conceptual representation of knowledge, have been used for text categorization (Scott and Matwin 1998).

5) ***Document Summarization***

WordNet has found useful application in text summarization. The approach presented by Barzilay and Elhadad (1997) utilizes information from WordNet to compute lexical chains.

12.3 FRAMENET

FrameNet[4] is a large database of semantically annotated English sentences. It is based on principles of frame semantics. It defines a tagset of semantic roles called the frame element. Sentences from the British National Corpus are tagged with these frame elements. The basic philosophy involved is that each word evokes a particular situation with particular participants. FrameNet aims at capturing these situations through case-frame representation of words (verbs, adjectives, and nouns). The word that invokes a frame is called *target word* or *predicate*, and the participant entities are defined using semantic roles, which are called *frame elements*. The FrameNet ontology can be viewed as a semantic level representation of predicate argument structure.

Each frame contains a main lexical item as predicate and associated frame-specific semantic roles, such as AUTHORITIES, TIME, and SUSPECT in the ARREST frame, called frame elements. As an example, consider sentence (12.1) annotated with the semantic roles AUTHORITIES and SUSPECT. The target word in sentence (12.1) is 'nab' which is a verb in the ARREST frame.

[$_{\text{Authorities}}$ The police] nabbed [$_{\text{suspect}}$ the snatcher]. (12.1)

A COMMUNICATON frame has the semantic roles ADDRESSEE, COMMUNICATOR, TOPIC, and MEDIUM. Figure 12.9 shows the core and non-core frame elements of the COMMUNICATION frame, along with other details. A JUDGEMENT frame contains roles such as JUDGE, EVALUEE, and REASON. A frame may inherit roles from another frame. For example, a STATEMENT frame may inherit from a COMMUNICATION frame; it contains roles such as SPEAKER, ADDRESSEE, and MESSAGE. The following sentences show some of these roles:

[$_{\text{Judge}}$ She] [$_{\text{Evaluee}}$ blames the police] [$_{\text{Reason}}$ for failing to provide enough protection]. (12.2)

[$_{\text{Speaker}}$ She] told [$_{\text{Addressee}}$ me] [$_{\text{Message}}$ 'I'll return by 7:00 pm today']. (12.3)

[4]http://framenet.icsi.berkeley.edu/

12.3.1 FrameNet Applications

Gildea and Jurafsky (2002) and Kwon et al. (2004) used FrameNet data for automatic semantic parsing. The shallow semantic role obtained from FrameNet can play an important role in information extraction. For example, a semantic role makes it possible to identify that the theme role played by 'match' is same in sentences (12.4) and (12.5) though the syntactic role is different.

The umpire stopped the match. (12.4)

The match stopped due to bad weather. (12.5)

In sentence (12.4), the word 'match' is the object, while it is the subject in sentence (12.5).

Semantic roles may help in the question-answering system. For example, the verb 'send' and 'receive' would share the semantic roles SENDER, RECIPIENT, GOODS, etc., (Gildea and Jurafsky 2002) when defined with respect to a common TRANSFER frame. Such common frames allow a question-answering system to answer a question such as 'Who sent packet to Khushbu?' using sentence (12.6).

Khushbu received a packet from the examination cell. (12.6)

Other applications include IR (Mohit and Narayanan 2003), interlingua for machine translation, text summarization, and word sense disambiguation.

Communication

Frame Elements

Core:

Addressee [Add]	Receiver of Message from the Communicator.
Communicator [Com]	The person conveying (written or spoken) a message to another person.
Message [Msg]	A proposition or set of propositions that the Communicator wants the Addressee to convey
Topic [Top]	The entity that the proposition(s) are about.

Non-core:

Amount_of_information [Amo]	The amount of information exchanged when communication occurs.
Depictive [Dep-Act]	The Depictive describes the state of the Communicator.
Duration []	The length of time during which the communication takes place.
Manner [Manr]	The Manner in which the Communicator communicates.
Means [Mns]	The Means by which the Communicator communicates.
Medium [Medium]	The physical or abstract setting in which the Message is conveyed.
Time []	The time at which the communication takes place.

Inherits From:
Is Inherited By: Communication_noise, Statement
Subframe of:
Has Subframes:
Uses: Topic
Is Used By: Claim_ownership, Communication_response, Contacting, Deny_permission, Discussion, Hear, Questioning, Reasoning, Reporting, Request, etc
Is Inchoative of:
Is Causative of:
See Also:

Sample Predicates

communicate, indicate, signal, speech

Figure 12.9 Frame elements of communication frame

12.4 STEMMERS

As discussed in Chapter 3, stemming, often called conflation, is the process of reducing inflected (or sometimes derived) words to their base or root form. The stem need not be identical to the morphological base of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root. Stemming is useful in search engines for query expansion or indexing and other NLP problems. Stemming programs are commonly referred to as stemmers. The most common algorithm for stemming English is Porter's algorithm[5] (Porter 1980). Other existing stemmers include Lovins[6] stemmer (Lovins 1968) and a more recent one called the Paice/Husk stemmer[7] (Paice 1990). Figure 12.10 shows a sample text and output produced using these stemmers.

Input Text:
Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation.

Output:
Lovins stemmer: such an analys can rev feature that ar not eas vis from the vari in th individu gen and can lead to a picture of expres that is mor biolog transpar and acces to interpres
Porter's stemmer: such an analysi can reveal feature that ar not easily visible from the variat in the individu gene and can lead to a picture of express that is more biolog transpar and access to interpret
Paice stemmer: Such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

Figure 12.10 Stemmed text using different stemmers

12.4.1 Stemmers for European Languages

There are many stemmers available for English and other languages. Snowball[8] presents stemmers for English, Russian, and a number of other European languages, including French, Spanish, Portuguese, Hungarian, Italian, German, Dutch, Swedish, Norwegian, Danish, and Finnish. The links for stemming algorithms for these languages can be found at *http://snowball.tartarus.org/texts/stemmersoverview.html.*

[5]http://tartarus.org/~martin/PorterStemmer/
[6]http://sourceforge.net/project/showfiles.php?group_id=24200
[7]http://www.comp.lancs.ac.uk/computing/research/stemming/Links/implementations.htm
[8]http://snowball.tartarus.org/

12.4.2 Stemmers for Indian Languages

Standard stemmers are not yet available for Hindi and other Indian languages. The major research on Hindi stemming has been accomplished by Ramanathan and Rao (2003) and Majumder et al. (2007). Ramanathan and Rao (2003) based their work on the use of handcrafted suffix lists. Majumder et al. (2007) used a cluster-based approach to find classes of root words and their morphological variants. They used a task-based evaluation of their approach and reported that stemming improves recall for Indian languages. Their observation on Indian languages was based on a Bengali data set. The Resource Centre of Indian Language Technology (CFILT), IIT Bombay has also developed stemmers for Indian languages, which are available at *http://www.cfilt.iitb.ac.in.*

12.4.3 Stemming Applications

Stemmers are common elements in search and retrieval systems such as Web search engines. Stemming reduces the variants of a word to same stem. This reduces the size of the index and also helps retrieve documents that contain variants of a query terms. For example, a user issuing a query for documents on 'astronauts' would like documents on 'astronaut' as well. Stemming permits this by reducing both versions of the word to the same stem. However, the effectiveness of stemming for English query systems is not too great, and in some cases may even reduce precision.

Text summarization and text categorization also involve term frequency analysis to find features. In this analysis, stemming is used to transform various morphological forms of words into their stems.

12.5 PART-OF-SPEECH TAGGER

Part-of-speech tagging is used at an early stage of text processing in many NLP applications such as speech synthesis, machine translation, IR, and information extraction. In IR, part-of-speech tagging can be used in indexing (for identifying useful tokens like nouns), extracting phrases and for disambiguating word senses. The rest of this section presents a number of part-of-speech taggers that are already in place.

12.5.1 Stanford Log-linear Part-of-Speech (POS) Tagger

This POS Tagger is based on maximum entropy Markov models. The key features of the tagger are as follows:

(i) It makes explicit use of both the preceding and following tag contexts via a dependency network representation.

Maximum Entropy Tagger (MET)

This tagger is based on a framework suggested by Ratnaparkhi (1997). It uses an iterative procedure to successively improve parameters for a set of features that help to distinguish between relevant contexts.

Trigram Tagger (T3)

This tagger is based on HMM. The states in the model are tag pairs that emit words. The technique has been suggested by Rabiner (1990) and the implementation is influenced by Brants (2000).

Error-driven Transformation-based Tagger (TBT)

This tagger is based on the transformation-based tagging approach proposed by Brill (1993). It uses annotated corpuses to learn transformation rules, which are then used to change the assigned tag using contextual information.

Example-based Tagger (ET)

The underlying assumption of example-based models (also called memory-based, instance-based or distance-based models) is that cognitive behaviour can be achieved by looking at past experiences that match the current problem, instead of learning and applying abstract rules. This framework has been suggested for NLP by Daelemans et al. (1996).

12.5.7 POS Tagger for Indian Languages

The automatic text processing of Hindi and other Indian languages is constrained heavily due to lack of basic tools and large annotated corpuses. Research groups are now focusing on removing these bottlenecks. The work on the development of tools, techniques, and corpora is going on at several places such as CDAC, IIT Bombay, IIIT Hyderabad, University of Hyderabad, CIIL Mysore, and University of Lancaster. IIT Bombay is involved in the development of morphology analysers and part-of-speech taggers for Hindi and Marathi. Both these languages have rich morphological structures. Their approach is based on *bootstrapping on a small corpus tagged* by a rule-based tagger and then applying statistical techniques to train a machine. More information can be found at *http://ltrc.iiit.net* and *www.cse.iitb.ac.in*. Work on Urdu part-of-speech taggers has been reported by Hardie (2003) and Baker et al. (2004).

12.6 RESEARCH CORPORA

Research corpora have been developed for a number of NLP-related tasks. In the following section, we point out few of the available standard document collections for a variety of NLP-related tasks, along with their Internet links.

12.6.1 IR Test Collection

We have already provided a list of IR test document collection in Chapter 9. Glasgow University, UK, maintains a list of freely available IR test collections. Table 12.2 lists the sources of those and few more IR test collections.

LETOR (learning to rank) is a package of benchmark data sets released by Microsoft Research Asia. It consists of two datasets OHSUMED and TREC (TD2003 and TD2004). LETOR is packaged with extracted features for each query-document pair in the collection, baseline results of several state-of-the-art learning-to-rank algorithms on the data and evaluation tools. The data set is aimed at supporting future research in the area of learning ranking function for information retrieval.

Table 12.2 IR test collection

LETOR	*http://research.microsoft.com/users/tyliu/LETOR/*
LISA	
CACM	
CISI	
MEDLINE	*http://www.dcs.gla.ac.uk/idom/ir_resources/test_collections/*
Cranfield	
TIME	
ADI	

12.6.2 Summarization Data

Evaluating a text summarizing system requires existence of 'gold summaries'. DUC provides document collections with known extracts and abstracts, which are used for evaluating performance of summarization systems submitted at TREC conferences. Figure12.11 shows a sample document and its extract from DUC 2002 summarization data.

<DOC>
<DOCNO> AP880911-0016 </DOCNO>
<FILEID>AP-NR-09-11-88 0423EDT</FILEID>
<FIRST>r i BC-HurricaneGilbert 09-11 0339</FIRST>
<SECOND>BC-Hurricane Gilbert,0348</SECOND>
<HEAD>Hurricane Gilbert Heads Toward Dominican Coast</HEAD>
<BYLINE>By RUDDY GONZALEZ</BYLINE>
<BYLINE>Associated Press Writer</BYLINE>
<DATELINE>SANTO DOMINGO, Doeninican Republic (AP) </DATELINE>
<TEXT>

Hurricane Gilbert swept toward the Dominican Republic Sunday, and the Civil Defense alerted its heavily populated south coast to prepare for high winds, heavy rains, and high seas.

The storm was approaching from the southeast with sustained winds of 75 mph gusting to 92 mph.

"There is no need for alarm," Civil Defense Director Eugenio Cabral said in a television alert shortly before midnight Saturday.

Cabral said residents of the province of Barahona should closely follow Gilbert's movement. An estimated 100,000 people live in the province, including 70,000 in the city of Barahona, about 125 miles west of Santo Domingo.

Tropical Storm Gilbert formed in the eastern Caribbean and strengthened into a hurricane Saturday night. The National Hurricane Center in Miami reported its position at 2 a.m. Sunday at latitude 16.1 north, longitude 67.5 west, about 140 miles south of Ponce, Puerto Rico, and 200 miles southeast of Santo Domingo.

The National Weather Service in San Juan, Puerto Rico, said Gilbert was moving westward at 15 mph with a "broad area of cloudiness and heavy weather" rotating around the center of the storm.

The weather service issued a flash flood watch for Puerto Rico and the Virgin Islands until at least 6 p.m. Sunday.

Strong winds associated with the Gilbert brought coastal flooding, strong southeast winds and up to 12 feet to Puerto Rico's south coast. There were no reports of casualties.

San Juan, on the north coast, had heavy rains and gusts Saturday, but they subsided during the night.

On Saturday, Hurricane Florence was downgraded to a tropical storm and its remnants pushed inland from the U.S. Gulf Coast. Residents returned home, happy to find little damage from 80 mph winds and sheets of rain.

Florence, the sixth named storm of the 1988 Atlantic storm season, was the second hurricane. The first, Debby, reached minimal hurricane strength briefly before hitting the Mexican coast last month.

</TEXT>
</DOC>

Extract

Tropical Storm Gilbert in the eastern Caribbean strengthened into a hurricane Saturday night. The National Hurricane Center in Miami reported its position at 2 a.m. Sunday to be about 140 miles south of Puerto Rico and 200 miles southeast of Santo Domingo. It is moving westward at 15mph with a broad area of cloudiness and heavy weather with sustained winds of 75mph gusting to 92mph. The Dominican Republic's Civil Defense alerted that country's heavily populated south coast and the National Weather Service in San Juan, Puerto Rico issued a flood watch for Puerto Rico and the Virgin Islands until at least 6 p.m. Sunday.

Figure 12.11 Sample document from DUC 2002 and its extract

12.6.3 Word Sense Disambiguation

SEMCOR[12] is a sense-tagged corpus used in disambiguation. It is a subset of the Brown corpus, sense-tagged with WordNet synsets. Open Mind Word Expert[13] attempts to create a very large sense-tagged corpus. It collects word sense tagging from the general public over the Web.

12.6.4 Asian Language Corpora

The multilingual EMILLE corpus is the result of the enabling minority language engineering (EMILLE) project at Lancaster University, UK. The project focuses on generation of data, software resources and basic language engineering tools for the NLP of south Asian languages. Central Institute for Indian Languages (CIIL), the Indian partner in the project, extended the set of target languages to include a number of Indian languages. CIIL provides a wider range of data in these languages from a wide range of genres. The data sources that EMILLE made available include monolingual written and spoken corpuses, parallel and annotated corpuses. The full EMILLE/CIIL corpus is available for free, but for research use only, at the link *http://www.elda.org/catalogue/en/text/W0037.html*. Further details about the corpus can be found in the manual at the site *http://www.emille.lancs.ac.uk/manual.pdf*.

Corpus building in these languages is constrained by the scarcity of repositories of electronic text. The monolingual corpus includes written data for 14 South Asian languages and spoken data for five languages (Hindi, Bengali, Gujrati, Punjabi, and Urdu). The spoken corpus was constructed from radio broadcasts on the BBC Asia network. The parallel corpus contains English text and its translation in five languages. The text includes UK government advice leaflets which are published in multiple languages. The corpus is aligned at sentence level. The parallel corpus provided by EMILLE corpus is a valuable resource for statistical machine translation research. The annotated component includes Urdu data annotated for part-of-speech tagging, and a Hindi corpus annotated to show nature of demonstrative use.

12.7 JOURNALS AND CONFERENCES IN THE AREA

A wide number of conference proceedings and journals report research in the various areas of NLP. Most notable among them are those associated with Association for Computing Machinery (ACM), Association for

[12] http://www.cs.unt.edu/~rada/downloads.html#semcor
[13] http://teach-computers.org.

Computational Linguistics (ACL), its European counterpart EACL, Recherche d'Information Assistie par Ordinateur (RIAO) and the International Conferences on Computational Linguistics (COLING). The ACM SIGIR Conference is one of the major conferences held on research and development in information retrieval. It provides the international forum for dissemination of research and demonstration of new systems and techniques. The 30th Annual International ACM SIGIR Conference[14] was held on 23–27 July 2007 at Amsterdam. The Proceedings of Text Retrieval Conferences (TRECs)[15] are another important source of information. These proceedings report results from standardized evaluations organized by the US government. The TRECs have been organized regularly since 1992 as a part of the TIPSTER text retrieval. They were earlier known as the Document Understanding Conference or Message Understanding Conferences. The conference series is sponsored by the National Institute of Standards and Technology (NIST) with additional support from other US government agencies. The ACM Special Interest Group on Information Retrieval (ACM-SIGIR) focuses on IR related tasks, and ECIR is its European counterpart. The NTCIR (NII test collection for IR) focuses on information retrieval with Japanese and other Asian languages.

KES[16] International Conferences in Knowledge-Based and Intelligent Engineering & Information Systems have been a regular feature since 1997. The conference mainly focuses on applications of intelligent systems. The topics covered by KES includes general intelligent topics like neural networks, fuzzy techniques, genetic algorithms, knowledge representation and management, applications using intelligent techniques (e.g., speech processing and synthesis and NLP) and emerging intelligent technologies like intelligent information retrieval, intelligent web mining and applications, intelligent user interfaces, etc.

HLT-NACCL is sponsored by the North American chapter of the Association for Computational Linguistics.

The *Journal of Computational Linguistics* is a leading premier publication focussing on theoretical and linguistics aspects. More practical applications are covered in the *Natural Language Engineering Journal. Information Retrieval* by Kluwer, *Information Processing and Management* by Elsevier, ACM's *Transactions on Information Systems* (TOIS), *Journal of American Society for Information Sciences* are major journals covering a wide range of information

[14] http://www.sigir2007.org/
[15] http://trec.nist.gov/
[16] http://www.kesinternational.org/conferences.php

Research, International Journal of Information Technology and Decision Making (World Scientific), and *Journal of Digital Information Management and Information System.*

A few AI publications also report work on language processing. Among these are *Artificial Intelligence, Computational Intelligence*, IEEE's *Transaction on Intelligent Systems*, and *Journal of AI Research.*

SUMMARY

- Lexical resources such as WordNet and FrameNet can be used in a number of NLP-related tasks.
- Stemmers are useful in a number of information processing tasks such as information retrieval, text summarization, and text categorization.
- Widely known stemmers include Porter's and Lovins stemmers.
- Part-of-speech tagger is used to assign a part-of-speech, such as noun, verb, pronoun, preposition, adverb, and adjective, to each word in a sentence (or text).
- Taggers include stanford log-linear part-of-speech tagger, TnT, CLAWS, and Brill's tagger.
- TREC and SIGIR conferences offer useful resources for a number of information processing-related tasks.

REFERENCES

Ananthkrishnan, R. and Durgesh Rao, 2003, 'A lightweight stemmer for Hindi,' Workshop on Computational Linguistics for South Asian Languages, *The 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL'03)*, ACL, Morristown, NJ.

Baker, P., A. Hardie, A.M. McEnery, and B.D. Jayaram, 2004, 'Corpus linguistics and South Asian languages: corpus creation and tool development,' *Literary and Linguistic Computing*, 19(4), 509–24.

Barzilay, Regina and Michael Elhadad, 1997, 'Using lexical chains for text summarization,' *Proceedings of the Intelligent Scalable Text Summarization Workshop (ISTS'97)*, ACL, Madrid.

Brants, Thosrten, 2000, 'TnT–as statistical part-of-speech tagger,' *Proceedings of the Sixth Applied Natural Language Processing Conference (ANLP-2000)*, Seattle, WA.

Brill E., 1992, 'A simple rule-based part-of-speech tagger,' *Proceedings of the Third Conference on Applied Natural Language Processing*, ACL, Budapest, Hungary.