CHAPTER 1

INTRODUCTION

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.

1.1 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.

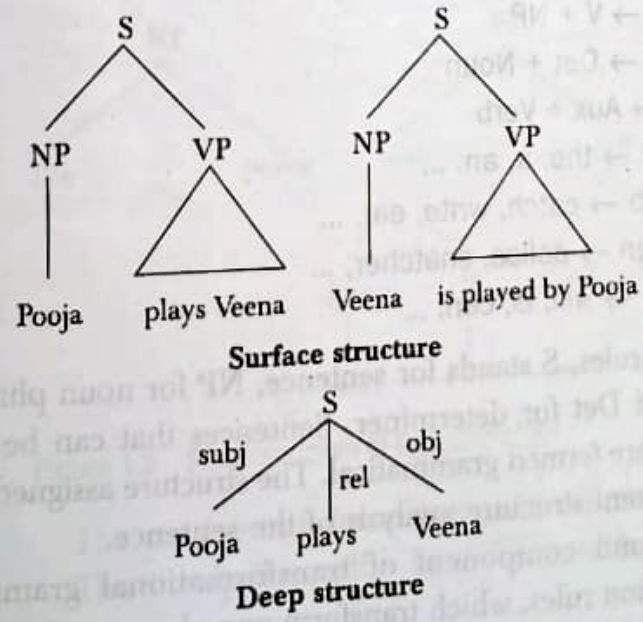


Figure 1.1 Surface and deep structures of sentence

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 (roles 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.

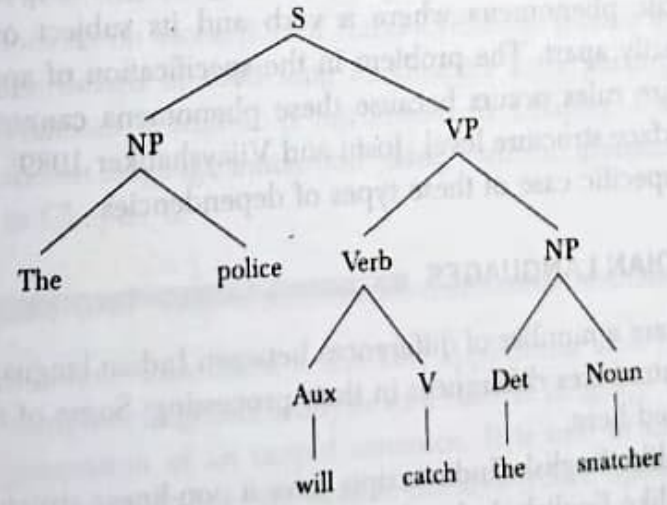


Figure 1.2 Parse structure of sentence (1.5)

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

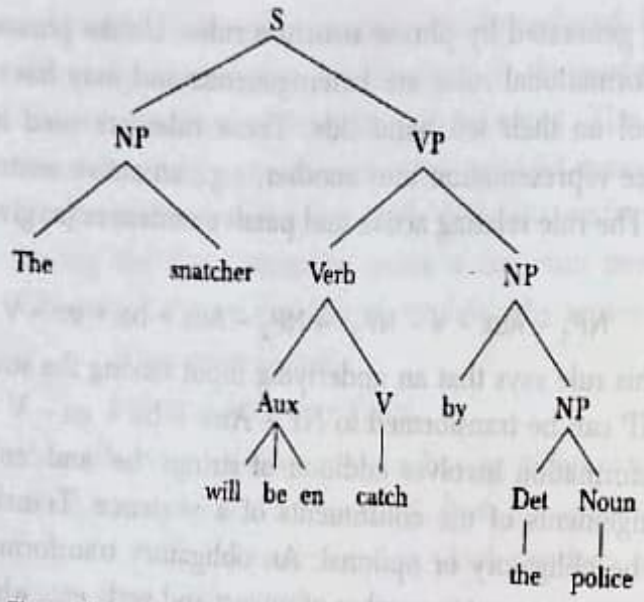


Figure 1.3 Structure of sentence (1.5) after applying passive transformations

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 a 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 rec 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* (Winograd 1972)**

This is a natural language understanding system that simulates actions 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 a 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

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 a 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.

2.2 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.

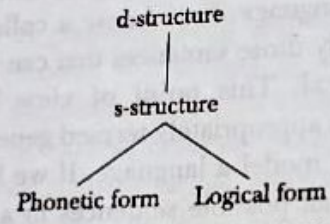


Figure 2.1 Different levels of representation in GB

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:

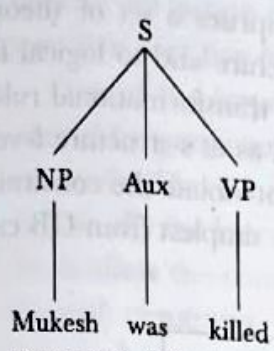


Figure 2.2 TG representation of sentence (2.1)

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

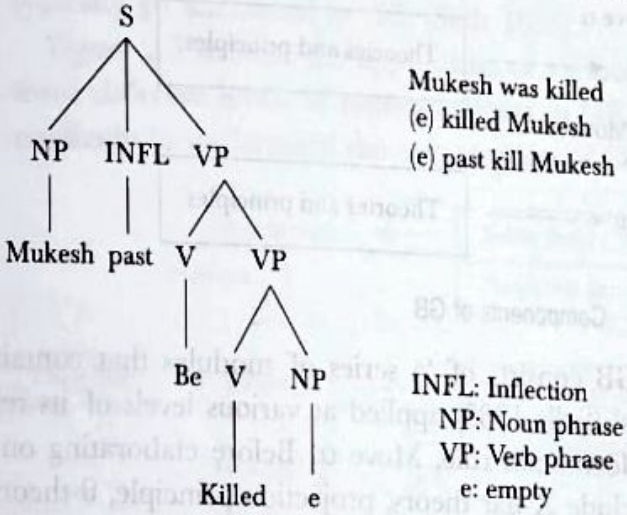


Figure 2.3 Surface structure of sentence (2.1) in GB

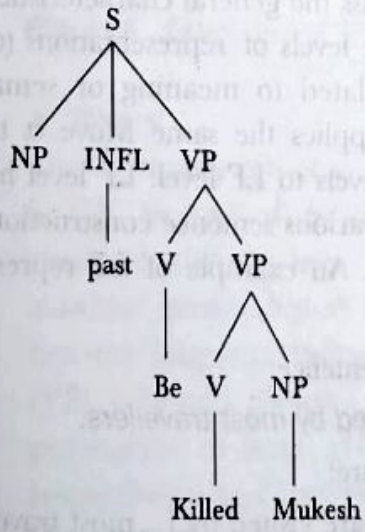


Figure 2.4 Deep structure of sentence (2.1) in GB

Components of GB

Government and binding (GB) comprises a set of theories that map 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 various theories and principles. Hence, in its simplest from GB can be represented by Figure 2.5.

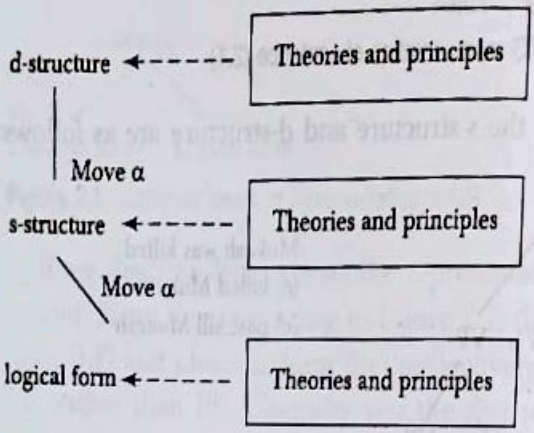


Figure 2.5 Components of GB

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.

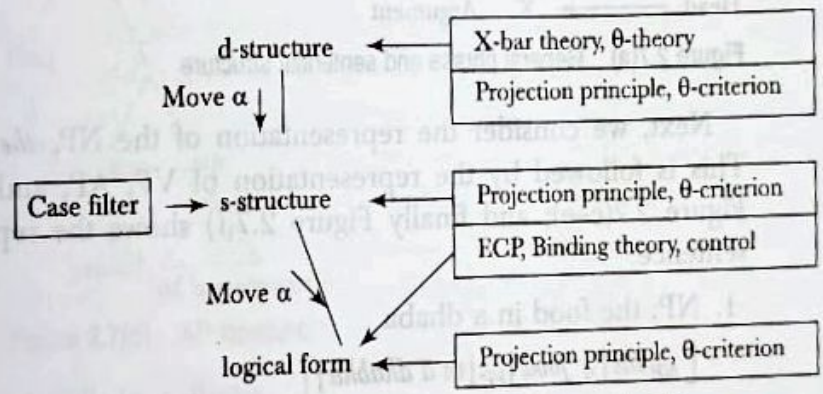


Figure 2.6 Organization of GB (adapted from Peter Sells 1985)

$\bar{X}$ *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).

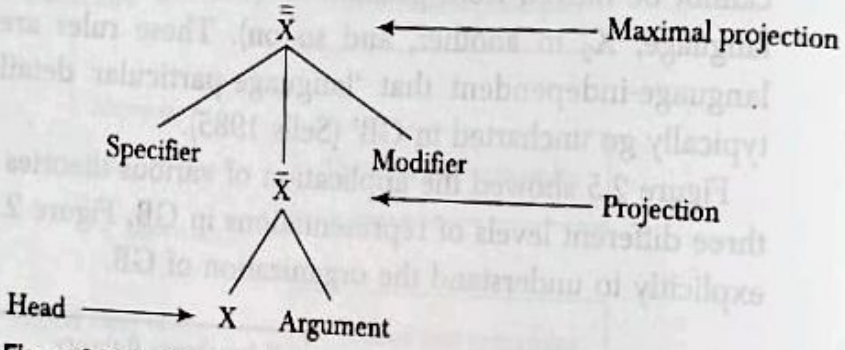


Figure 2.7(a) General phrase and sentential structure

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

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

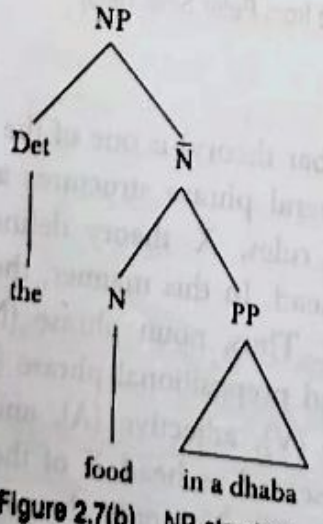


Figure 2.7(b) NP structure

2. VP: ate the food in a dhaba

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

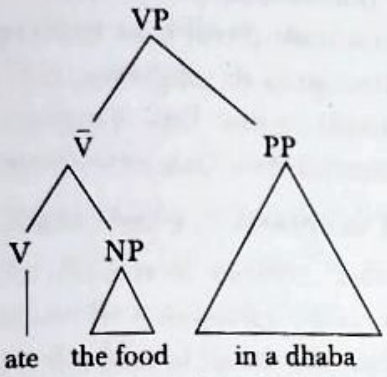


Figure 2.7(c) VP structure

3. AP: very proud of his country

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

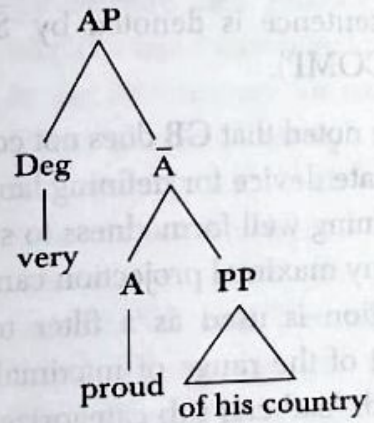


Figure 2.7(d) AP structure

4. PP: in a dhaba

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

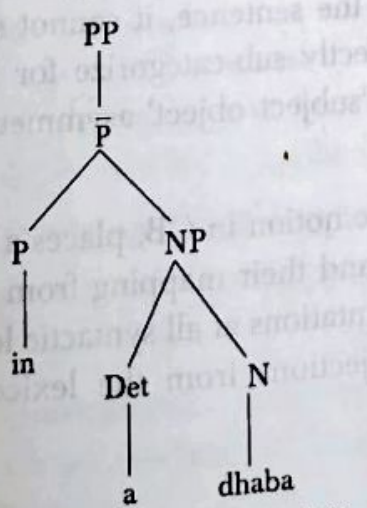


Figure 2.7(e) PP structure

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

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

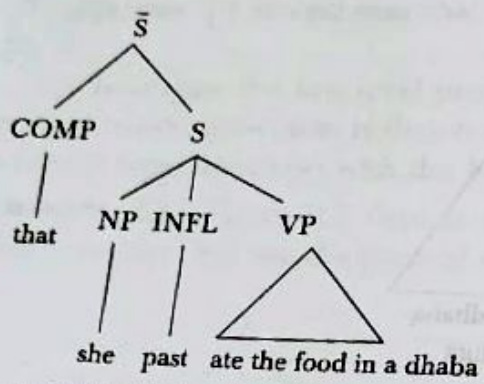


Figure 2.7(f) Maximal projection of sentence structure

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

---

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 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 ?
[Mukesh INFL eat what] (2.3)

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 phra and sentence structure, functional specifications are annotated on th nodes of c-structure, which when applied on sentences, results in f-structur Hence, f-structure is the final product which encodes the informatio obtained from phrase and sentence structure rules and functiona specifications.

Let us consider an example.

Example 2.5

She saw stars in the sky.

CFG rules to handle this sentence are:

$S \rightarrow NP\ VP$

$VP \rightarrow V\ \{NP\}\ \{NP\}\ PP^*\ \{S'\}$

$PP \rightarrow P\ NP$

$NP \rightarrow Det\ N\ \{PP\}$

$S' \rightarrow Comp\ S$

where

S: sentence
P: preposition
S′: clause
{ } optional
V: verb
N: noun
Comp: complement
*: Phrase can appear any number of times including blank.

When annotated with functional specifications, the rules become:

Rule 1: $S \rightarrow \underset{\uparrow \text{subj} = \downarrow}{NP}\ \underset{\uparrow = \downarrow}{VP}$

Rule 2: $VP \rightarrow \underset{\uparrow \text{obj} = \downarrow}{V}\ \underset{\uparrow \text{obj 2} = \downarrow}{\{NP\}}\ \underset{\uparrow (\downarrow \text{case}) = \downarrow}{\{NP\}}\ \underset{\uparrow \text{comp} = \downarrow}{PP^*}\ \{S'\}$

Rule 3: $PP \rightarrow \underset{\uparrow \text{obj} = \downarrow}{P}\ NP$

Rule 4: $NP \rightarrow \{Det\}\ N\ \underset{\uparrow \text{Adjunct} = \downarrow}{\{PP\}}$

Rule 5: $S' \rightarrow \text{Comp}\ \underset{\uparrow = \downarrow}{S}$

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-sructure of V can be obtained from the lexicon itself. All terminals in LFG can be thought of as annotated with $\uparrow = \downarrow$. 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. '$\uparrow$ ($\downarrow$case) = $\downarrow$' 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.

She saw stars. (2.4)

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

Saw V $\uparrow$ Pred = 'see < ($\uparrow$ Subj) ($\uparrow$ Obj) >'
($\uparrow$ Tense = PAST)

Stars N $\uparrow$ Pred = 'Star'
$\uparrow$ Pers = 3
$\uparrow$ Num = PL

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

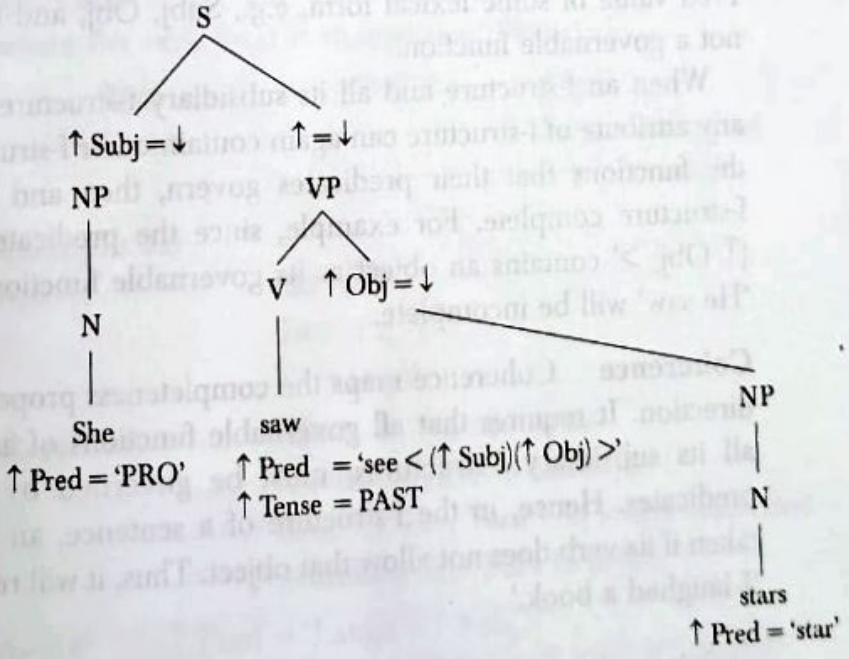


Figure 2.8 C-structure of sentence (2.4)

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

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

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 requ... new subject, while the old subject becomes the object of the new pred... and the old verb becomes the X complement (complement to infi... VPs).

Long Distance Dependencies and Coordination

In GB, when a category moved, it creates an empty category. In L... unbounded movement and coordination is handled by the functi... identity and by correlation with the corresponding f-structure. An exam... 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:

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

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

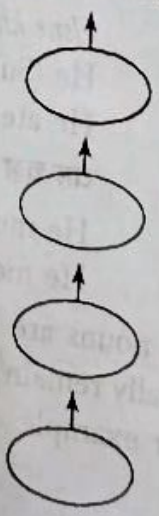
रेखा के पिता	उसके पिता
Rekha ke pita	*Uske pita*
Rekha of father	
Father of Rekha	Her (His) father

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

ताला खो गया	चाभी खो गयी
Taalaa kho gayaa	*Chaabhii kho gayeee*
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, surfa structure, and logical form (LF), where the LF is nearer to semantics. Th theory tries to resolve all language issues at syntactic levels only. Unli GB, Paninian grammar framework is said to be syntactico-semantic, th is, one can go from surface layer to deep semantics by passing throu intermediate layers. Although all these layers are not named, as pe 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).

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:

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

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

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 probability of a word, given the sequence of words preceding it. This 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.

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

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:

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

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

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.

$$P(\text{The/<s>}) \times P(\text{Arabian/the}) \times P(\text{Knights/Arabian}) \times P(\text{are/knights}) \times P(\text{the/are}) \times P(\text{fairy/the}) \times P(\text{tales/fairy}) \times P(\text{of/tales}) \times P(\text{the/of}) \times P(\text{east/the})$$
$$= 0.67 \times 0.5 \times 1.0 \times 1.0 \times 0.5 \times 0.2 \times 1.0 \times 1.0 \times 1.0 \times 0.2$$
$$= 0.0067$$

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. 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:

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

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

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 FSA 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, 2004'
[^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 lowercase '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 string containing either 's' or 'S'. The pattern /[sS]ana/ matches with the strings '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', 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 work. The pattern blackberry|ies matches with either 'blackberry' or This is because sequences take precedence over disjunction. In order apply the disjunction operator to a specific pattern, we need to enclose within parentheses. The parenthesis operator makes it possible to the enclosed pattern as a single character for the purposes of neighbor operators. Now, we will consider an example from real application.

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

3.3 FINITE-STA

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, it has 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 are 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 in 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.

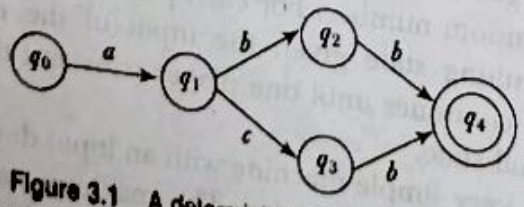


Figure 3.1 A deterministic finite-state automaton (DFA)

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.

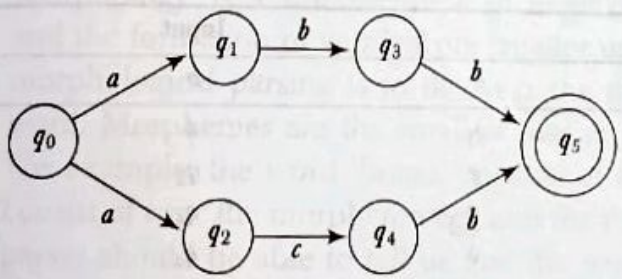


Figure 3.2 Non-deterministic finite-state automaton (NFA)

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 recog a string. The set of all strings that leave us in a final state is called language accepted or defined by the FA. This means *ac* is not a word the language defined by the automaton of Figure 3.1.

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

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

The example considered here is quite simple. Typically, the list o transition rules can be quite long. Listing all transition rules may b inconvenient, so often we represent an automaton as a *state-transitio 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	ϕ
q_2	ϕ	q_4	q_3
q_3	ϕ	q_4	ϕ
final: q_4	ϕ	ϕ	ϕ

Now, consider a language consisting of all strings containing only *a* 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.

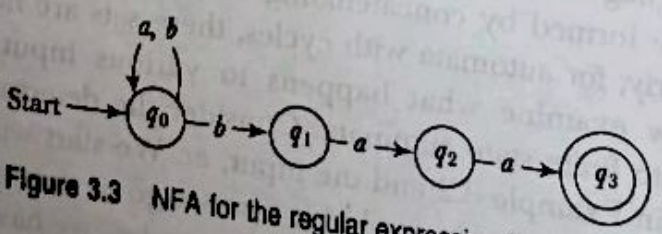


Figure 3.3 NFA for the regular expression /(a|b)*baa$/

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.

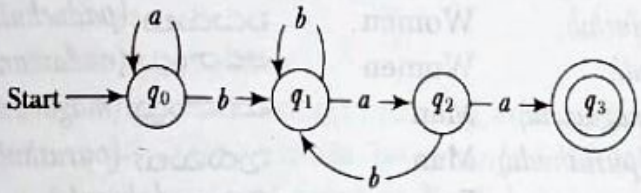


Figure 3.4 Equivalent DFA for NFA in Figure 3.3

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