PBR VITS (AUTONOMOUS)

IV B.TECH CSE-IOT

NATURAL LANGUAGE PROCESSING

TEXT BOOK: James Allen, **Natural Language Understanding**, 2nd Edition, 2003, Pearson Education

Course Instructor: Dr KV Subbaiah, M.Tech, Ph.D, Professor, Dept. of CSE

UNIT–I Introduction to Natural language

The Study of Language, Applications of NLP, Evaluating Language Understanding Systems, Different Levels of Language Analysis, Representations and Understanding, Organization of Natural language Understanding Systems, Linguistic Background: An outline of English Syntax.

1.1 Introduction to NLP:

As we all know Natural Language refers to the language spoken by people e.g. English, Hindi, Telugu as opposed to artificial/programming languages, like C, C++, Java, etc.

Natural Language Processing (NLP) is a field of research which determines the way computers can be used to understand and manage natural language text or speech to do useful things.

Natural Language Processing (NLP) a subset technique of Artificial Intelligence which is used to narrow the communication gap between the Computer and Human. It is originated from the idea of Machine Translation (MT) which came to existence during the 1950. The primary aim was to translate one human language to another human language, for example, Russian language to English language using the brain of the Computers but after that, the thought of conversion of human language to computer language and vice-versa emerged, so that communication with the machine became easy.

NLP is a process where input provided in a human language and converts this input into a useful form of representation. The field of NLP is primarily concerned with getting computers to perform interesting and useful tasks with human languages. The field of NLP is secondarily concerned with helping us come to a better understanding of human language.

As stated above the idea had emerged from the need for Machine Translation in the 1950s. Then the original language was English and Russian. But the use of other words such as Chinese also came into existence in the initial period of the 1960s.

In the 1960s, the NLP got a new life when the idea and need of Artificial Intelligence emerged.

In 1978 LUNAR is developed by W.A woods; it could analyze, compare and evaluate the chemical data on a lunar rock and soil composition that was accumulating as a result of Apollo moon missions and can answer the related question.

In the 1980s the area of computational grammar became a very active field of research which was linked with the science of reasoning for meaning and considering the user‘s beliefs and intentions.

In the period of 1990s, the pace of growth of NLP increased. Grammars, tools and Practical resources related to NLP became available with the parsers. Probabilistic and data-driven models had become quite standard by then.

In 2000, Engineers had a large amount of spoken and textual data available for creating systems. Today, large amount of work is being done in the field of NLP using Machine Learning or Deep Neural Networks in general, where we are able to create state-of-the-art models in text classification, Question and Answer generation, Sentiment Classification, etc.

1.2 GENERIC NLP SYSTEM

Natural language Processing should start with some input and ends with effective and accurate output. The possible inputs to an NLP are quite broad. Language can be in a variety of forms such as the paragraphs of text, commands that are typed directly to a computer system, etc. The input language might be given to the system a sentence at a time or it might be multiple sentences all at once. The inputs for natural language processor can be typed input, message text or speech. NLP systems have some kind of pre-processor. Data preprocessing involves preparing and cleaning text data for machines to be able to analyze it. Preprocessing puts data in workable form and highlights features in the text that an algorithm can work with. Preprocessing does dictionary lookup, morphological analysis, lexical substitutions, and part-of-speech assignment.

There are variety of output can be generated by the system. The output from a system that incorporates an NLP might be an answer from a database, a command to change some data in a database, a spoken response, Semantics, Part of speech, Morphology of word, Semantics of the word or some other action on the part of the system. Remember these are the output of the system as a whole, not the output of the NLP component of the system.

Figure 1.2.1 shows a generic NLP system and its input-output variety. Figure 1.2.2 shows a typical view of what might be inside the NLP box of Figure 1.2.1. Each of the boxes in Figure 1.2.2 represents one of the types of processing that make up an NLP analysis.

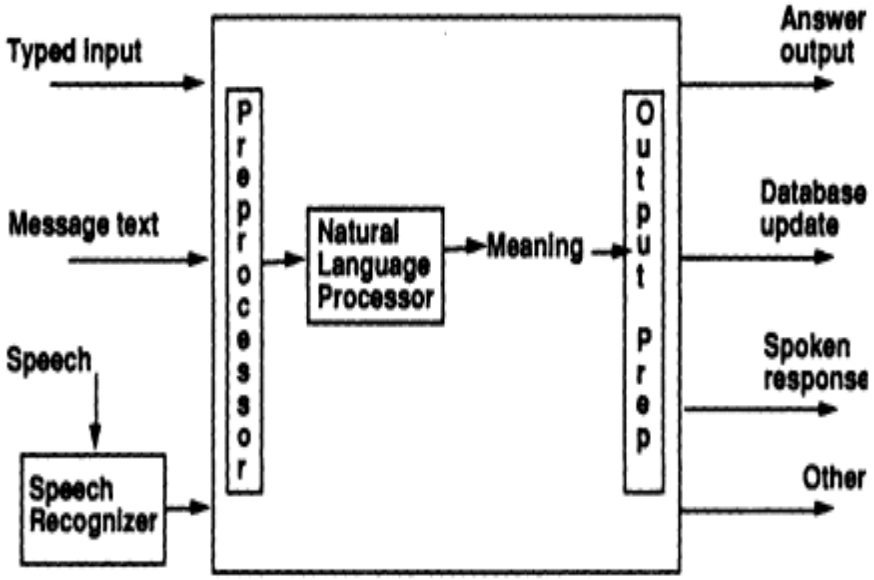


Figure 1.2.1 Generic NLP system.

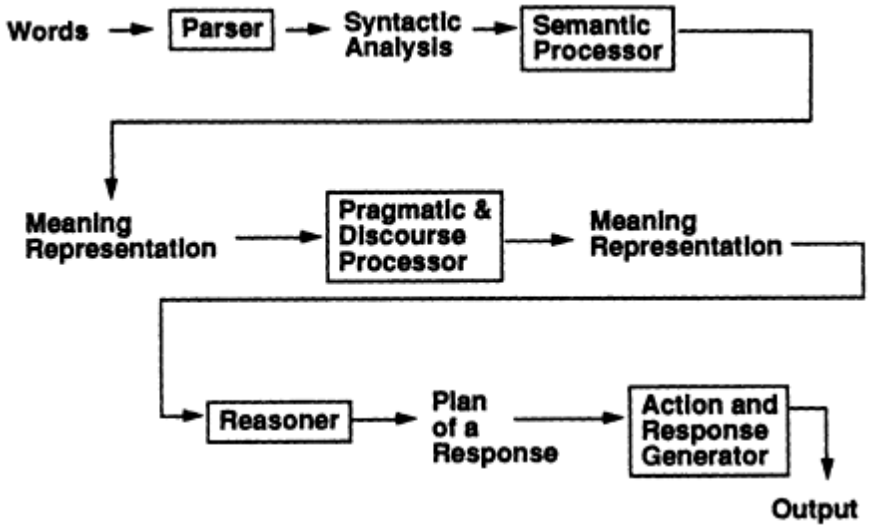


Figure 1.2.2 Pipeline view of the components of a generic NLP system.

1.3 LEVELS OF NLP

There are two components of Natural Language Processing

I. Natural Language Understanding (NLU)

• NLU takes some spoken/typed sentence and working out what it means.

• Here different level of analysis required such as morphological analysis, syntactic analysis, semantic analysis, discourse analysis, …

II Natural Language Generation (NLG)

• NLG takes some formal representation of what you want to say and working out a way to express it in a natural (human) language (e.g., English)

• Here different level of synthesis required: deep planning (what to say), syntactic generation .

Difference between NLU and NLG

NLU	NLG
It is the process of reading and interpreting language.	It is the process of writing or generating language.
NLU explains the meaning behind the written text or speech in natural language	NLG generates the natural language using machines.
NLU draws facts from the natural language using various tools and technologies such as parsers, POS taggers, etc,	NLG uses the insights generated from parsers, POS tags, etc. to generate the natural language.
NLU understands the human language and converts it into data	NLG uses the structured data and generates meaningful narratives out of it

The NLP can broadly be divided into various levels as shown in Figure 1.3.1.

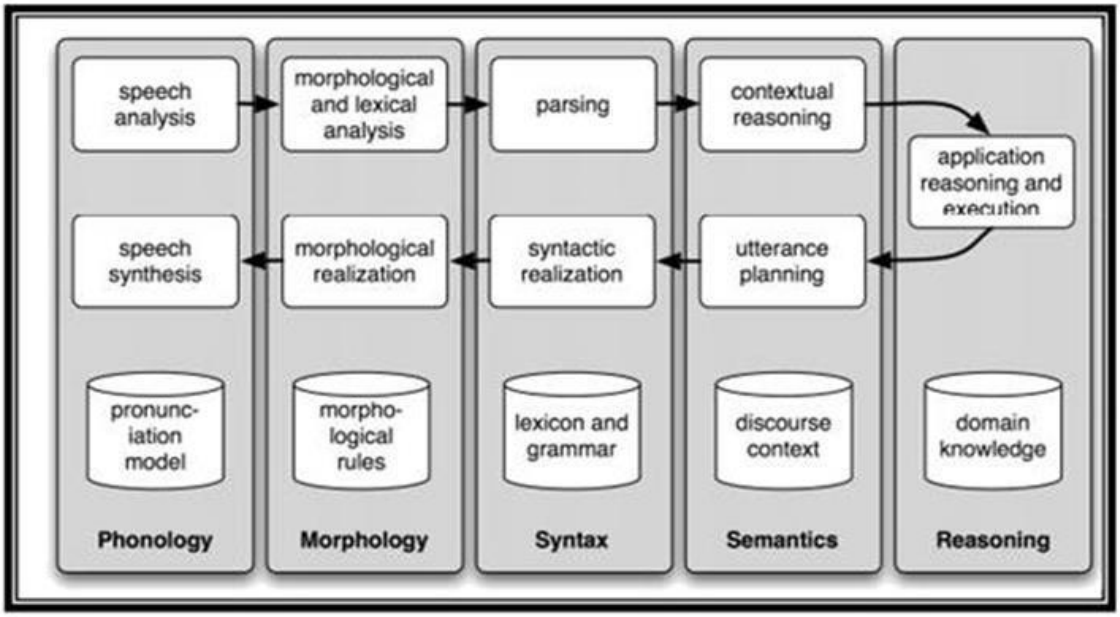


Figure 1.3.1 Levels of NLP

1. Phonology:
It concerned with interpretation of speech sound within and across words.

2. Morphology:
It deals with how words are constructed from more basic meaning units called morphemes. A morpheme is the primitive unit of meaning in a language. For example, "truth+ful+ness".

3. Syntax:
It concerns how words can be put together to form correct sentences and determines what structural role each word plays in the sentence and what phrases are subparts of other phrases.
For example, "the dog ate my homework"

4. Semantics:
It is a study of the meaning of words and how these meaning combine in sentences to form sentence meaning. It is study of context- independent meaning. For example, plant: industrial plant/ living organism . Pragmatics concerns with how sentences are used in different situations and how it affects the interpretation of the sentence. Discourse context deals with how the immediately preceding sentences affect the interpretation of the next sentence. For example, interpreting pronouns and interpreting the temporal aspects of the information.

5. Reasoning:
To produce an answer to a question which is not explicitly stored in a database; Natural Language Interface to Database (NLIDB) carries out reasoning based on data stored in the database. For example, consider the database that holds the student academic information, and user posed a query such as: 'Which student is likely to fail in Science subject?' To answer the query, NLIDB needs a domain expert to narrow down the reasoning process.

1.4 The Study of Language:

Language is studied in several different academic disciplines. Each discipline defines its own set of problems and has its own methods for addressing them. The linguist, for instance, studies the structure of language itself, considering questions such as why certain combinations of words form sentences but others do not, and why a sentence can have some meanings but not others. The psycholinguist, on the other hand, studies the processes of human language production and comprehension, considering questions such as how people identify the appropriate structure of a sentence and when they decide on the appropriate meaning for words. The philosopher considers how words can mean anything at all and how they identify objects in the world. Philosophers also consider what it means to have beliefs, goals, and intentions, and how these cognitive capabilities relate to language. The goal of the computational linguist is to develop a compu tational theory of language, using the notions of algorithms and data structures from computer science. Of course, to build a computational model, you must take advantage of what is known from all the other disciplines.

Table 1.1 summarizes these different approaches to studying language.

Discipline	Typical Problems	Tools
Linguists	How do words form phrases and sentences? What constrains the possible meanings for a sentence?	Intuitions about wellformedness and meaning; mathematical models of structure (for example, formal language theory,model theoretic semantics)
Psycholinguists	How do people identify the structure of sentences? How are word meanings identified? When does understanding take place?	Experimental techniques based on measuring human performance; statistical analysis of observations
Philosophers	What is meaning, and how do words and sentences acquire it? How do words identify objects in the world?	Natural language argumentation using intuition about counter-examples; mathematical models (for example, logic and model theory)
Computational Linguists	How is the structure of sentences identified? How can knowledge and reasoning be modeled? How can language be used to accomplish specific tasks?	Algorithms, data structures; formal models of representation and reasoning; AI techniques (search and representation methods)

Table 1.1 The major disciplines studying language

1.5 Applications of NLP:

a) Machine Translation

Machine translation is basically used to convert text or speech from one natural language to another natural language. Machine translation, an integral part of Natural Language Processing where translation is done from source language to target language preserving the meaning of the sentence. Example: Google Translator

b)Information Retrieval

It refers to the human-computer interaction (HCI) that happens when we use a machine to search a body of information for information objects (content) that match our search query. A Person's query is matched against a set of documents to find a subset of 'relevant' document. Examples: Google, Yahoo, Altavista, etc.

c) Text Categorization

Text categorization (also known as text classification or topic spotting) is the task of automatically sorting a set of documents into categories (clusters).

Uses of Text Categorization

- Filtering of content
- Spam filtering Identification of document content
- Survey coding

d) Information Extraction

Identify specific pieces of information in unstructured or semi-structured textual document. Transform unstructured information in a corpus of documents or web pages into a structured database.

Applied to different types of text:

- Newspaper articles
- Web pages
- Scientific articles
- Newsgroup messages
- Classified ads
- Medical notes

e)Grammar Correction-

In word processor software like MS-word, NLP techniques are widely used for spelling correction & grammar check.

f) Sentiment Analysis-

Sentiment Analysis is also known as opinion mining. It is mainly used on the web to analyse the behaviour, attitude, and emotional state of the sender. This application is implemented through a combination of NLP) and statistics by assigning the values to the text (natural, positive or negative), identify the mood of the context (sad, happy, angry, etc.)

g)Question-Answering systems-
Question Answering focuses on constructing systems that automatically answer the questions asked by humans in a natural language. It presents only the requested information instead of searching full documents like search engine. The basic idea behind the QA system is that the users just have to ask the question and the system will retrieve the most appropriate and correct answer for that question.
E.g.
Q. "What is the birth place of Shree Krishna?
A. Mathura

h)Spam Detection
To detect unwanted e-mails getting to a user's inbox, spam detection is used

i)Chatbot
Chatbot is one of the most important applications of NLP. It is used by many companies to provide the customer's chat services.

j) Speech Recognition-
Speech recognition is used for converting spoken words into text. It is used in applications, such as mobile, home automation, video recovery, dictating to Microsoft Word, voice biometrics, voice user interface, and so on.

k)Text summarization
This task aims to create short summaries of longer documents while retaining the core content and preserving the overall meaning of the text.

1.6 Evaluating Language Understanding Systems

Natural language understanding (NLU) Natural language understanding is a branch of artificial intelligence that uses computer software to understand input in the form of sentences using text or speech.

NLU enables human-computer interaction. It is the comprehension of human language such as English, Hindi, Telugu, Spanish and French, for example, that allows computers to understand commands without the formalized syntax of computer languages. NLU also enables computers to communicate back to humans in their own languages.
The main purpose of NLU is to create chat- and voice-enabled bots that can interact with the public without supervision. Many major IT companies, such as Amazon, Apple, Google and Microsoft, and startups have NLU projects underway.

How does natural language understanding work?

NLU analyzes data to determine its meaning by using algorithms to reduce human speech into a structured ontology -- a data model consisting of semantics and pragmatics definitions. Two fundamental concepts of NLU are intent and entity recognition.
Intent recognition is the process of identifying the user's sentiment in input text and determining their objective. It is the first and most important part of NLU because it establishes the meaning of the text.

Entity recognition is a specific type of NLU that focuses on identifying the entities in a message, then extracting the most important information about those entities. There are two types of entities: named entities and numeric entities. Named entities are grouped into categories -- such as people, companies and locations. Numeric entities are recognized as numbers, currencies and percentages.
For example, a request for an island camping trip on Vancouver Island on Aug. 18 might be broken down like this: ferry tickets [intent] / need: camping lot reservation [intent] / Vancouver Island [location] / Aug. 18 [date].

Natural language understanding Systems

Here are examples of applications that are designed to understand language as humans do, rather than as a list of keywords. NLU is the basis of speech recognition software -- such as Siri on iOS -- that works toward achieving human-computer understanding.

IVR and message routing. Interactive Voice Response (IVR) is used for self-service and call routing. Early iterations were strictly touchtone and did not involve AI. However, as IVR technology advanced, features such as NLP and NLU have broadened its capabilities and users can interact with the phone system via voice. The system processes the user's voice, converts the words to text, and then parses the grammatical structure of the sentence to determine the probable intent of the caller.

Customer support and service through intelligent personal assistants. NLU is the technology behind chatbots, which is a computer program that converses with a human in natural language via text or voice. Chatbots follow a script and can only answer questions in that script. These intelligent personal assistants can be a useful addition to customer service. For example, chatbots are used to provide answers to frequently asked questions. Accomplishing this involves layers of different processes in NLU technology, such as feature extraction and classification, entity linking and knowledge management.

Machine translation. Machine learning (ML) is a branch of AI that enables computers to learn and change behavior based on training data. Machine learning algorithms are also used to generate natural language text from scratch. In the case of translation, a machine learning algorithm analyzes millions of pages of text -- say, contracts or financial documents -- to learn how to translate them into another language. The more documents it analyzes, the more accurate the translation. For example, if a user is translating data with an automatic language tool such as a dictionary, it will perform a word-for-word substitution. However, when using machine translation, it will look up the words in context, which helps return a more accurate translation.

Data capture. Data capture is the process of gathering and recording information about an object, person or event. For example, if an e-commerce company used NLU, it could ask customers to enter their shipping and billing information verbally. The software would understand what the customer meant and enter the information automatically.

Conversational interfaces. Many voice-activated devices -- including Amazon Alexa and Google Home -- allow users to speak naturally. By using NLU, conversational interfaces can understand and respond to human language by segmenting words and sentences, recognizing grammar, and using semantic knowledge to infer intent.

1.7 The Different Levels of Language Analysis

A natural language-system must use considerable knowledge about the structure of the language itself, including what the words are, how words combine to form sentences, what the words mean, how word meanings contribute to sentence meanings, and so on. However, we cannot completely account for linguistic behavior without also taking into account another aspect of what makes humans intelligent — their general world knowledge and their reasoning abilities. For example, to answer questions or to participate in a conversation, a person not only must know a lot about the structure of the language being used, but also must know about the world in general and the conversational setting in particular.

The following are some of the different forms of knowledge relevant for natural language understanding:

Phonetic and phonological knowledge - concerns how words are related to the sounds that realize them. Such knowledge is crucial for speech-based systems.

Morphological knowledge - concerns how words are constructed from more basic meaning units called morphemes. A morpheme is the primitive unit of meaning in a language (for example, the meaning of the word "*friendly"* is derivable from the meaning of the noun "*friend"* and the suffix "*-ly",* which transforms a noun into an adjective).

Syntactic knowledge - concerns how words can be put together to form correct sentences and determines what structural role each word plays in the sentence and what phrases are subparts of what other phrases.

Semantic knowledge - concerns what words mean and how these meanings -combine in sentences to form sentence meanings. This is the study of context-independent meaning - the meaning a sentence has regardless of the context in which it is used.

Pragmatic knowledge - concerns how sentences are used in different situations and how use affects the interpretation of the sentence.

Discourse knowledge-concerns how the immediately preceding sentences affect the interpretation of the next sentence. This information is especially important for interpreting pronouns and for interpreting the temporal aspects of the information conveyed.

World knowledge - includes the general knowledge about the structure of the world that language users must have in order to, for example, maintain a conversation. It includes what each language user must know about the other user's beliefs and goals.

1.8 Representation and Understanding

Ambiguity can occur at all NLP levels. It is a property of linguistic expressions. If an expression (word/phrase/sentence) has more than one meaning we can refer it as ambiguous. To represent meaning, we must have a more precise language. The tools to do this come from mathematics and logic and involve the use of formally specified representation languages. Formal languages are specified from very simple building blocks. The most fundamental is the notion of an atomic symbol which is distinguishable from any other atomic symbol simply based on how it is written. Useful representation languages have the following two properties:

● The representation must be precise and unambiguous. You should be able to express every distinct reading of a sentence as a distinct formula in the representation.

● The representation should capture the intuitive structure of the natural language sentences that it represents. For example, sentences that appear to be structurally similar should have similar structural representations, and the meanings of two sentences that are paraphrases of each other should be closely related to each other.

Syntax: Representing Sentence Structure

The syntactic structure of a sentence indicates the way that words in the sentence are related to each other. This structure indicates how the words are grouped together into phrases, what words modify what other words, and what words are of central importance in the sentence. In addition, this structure may identify the types of relationships that exist between phrases and can store other information about the particular sentence structure that may be needed for later processing. For example, consider the following sentences:

1. John sold the book to Mary.

2. The book was sold to Mary by John.

These sentences share certain structural properties. In each, the noun phrases are "John", "Mary", and "the book", and the act described is some selling action. In other respects, these sentences are significantly different. For instance, even though both sentences are always either true or false in the exact same situations, you could only give sentence 1 as an answer to the question "What did John do for Mary?" Sentence 2 is a much better continuation of a sentence beginning with the phrase "After it fell in the river", as sentences 3 and 4 show. Following the standard convention in linguistics, this book will use an asterisk (*) before any example of an ill-formed or questionable sentence.

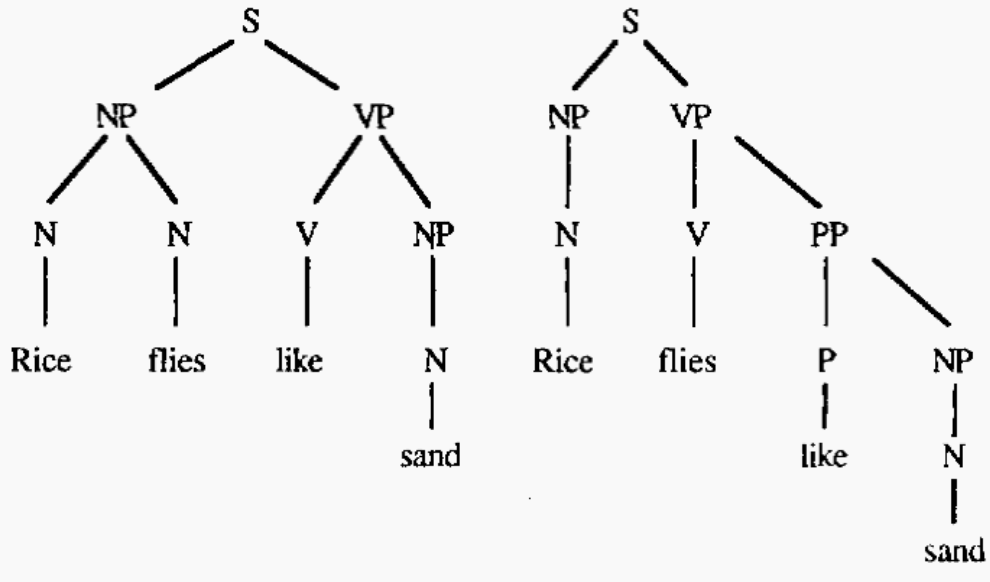


Figure 1.4 Two structural representations of *Rice flies like sand.*

Most syntactic representations of language are based on the notion of context-free grammars, which represent sentence structure in terms of what phrases are subparts of other phrases. This information is often presented in a tree form, such as the one shown in Figure 1.4, which shows two different structures for the sentence "*Rice flies like sand"*. In the first reading, the sentence is formed from a noun phrase (NP) describing a type of fly' rice flies, and a verb phrase (VP) that asserts that these flies like sand. In the second structure, the sentence is formed from a noun phrase describing a type of substance, rice, and a verb phrase stating that this substance flies like sand (say, if you throw it). The two structures also give further details on the structure of the noun phrase and verb phrase and identify the part of speech for each word. In particular, the word "*like"* is a verb (V) in the first reading and a preposition (P) in the second.

3. *After it fell in the river, John sold Mary the book.

4. After it fell in the river, the book was sold to Mary by John.

Many other structural properties can be revealed by considering sentences that are not well-formed. Sentence 5 is ill-formed because the subject and the verb do not agree in number (the subject is singular and the verb is plural), while 6 is ill-formed because the verb *put* requires some modifier that describes where John put the object.

5. *John are in the corner.

6. *John put the book.

The Logical Form

The structure of a sentence doesn't reflect its meaning, however. For example, the NP "*the catch"* can have different meanings depending on whether the speaker is talking about a baseball game or a fishing expedition. Both these inter-pretations have the same syntactic structure, and the different meanings arise from an ambiguity concerning the sense of the word "*catch"*. Once the correct sense is identified, say the fishing sense, there still is a problem in determining what fish are being referred to. The intended meaning of a sentence depends on the situation in which the sentence is produced.

The Final Meaning Representation

The final representation needed is a general knowledge representation (KR), which the system uses to represent and reason about its application domain. This is the language in which all the specific knowledge based on the application is represented. The goal of contextual interpretation is to take a representation of the structure of a sentence and its logical form, and to map this into some expression in the KR that allows the system to perform the appropriate task in the domain. In a question-answering application, a question might map to a database query, in a story-understanding application, a sentence might map into a set of expressions that represent the situation that the sentence describes.

1.9 The Organization of Natural Language Understanding Systems

There are general five steps in natural language processing

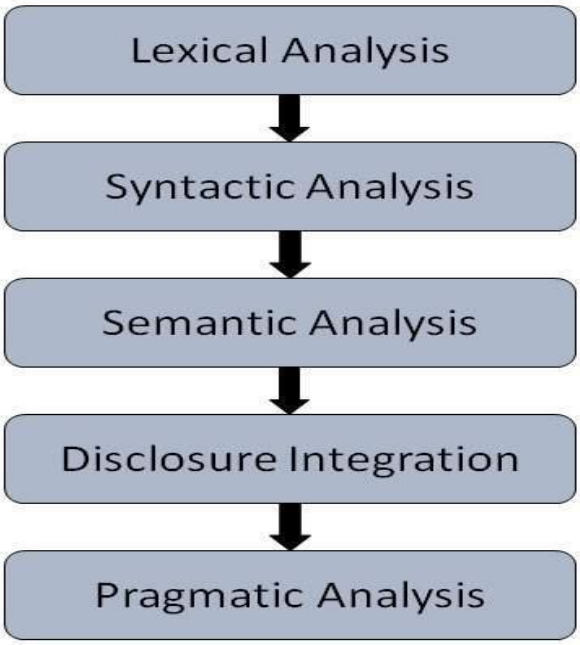


Figure 1.6.1 General five steps of NLP

1) Lexical Analysis:

It is the first stage in NLP. It is also known as morphological analysis. It consists of identifying and analyzing the structure of words. Lexicon of a language means the collection of phrases and words in a language. Lexical analysis is dividing the whole chunk of text into words, sentences, and paragraphs.

2) Syntactic Analysis:

Syntactic analysis consists of analysis of words in the sentence for grammar and ordering words in a way that shows the relationship among the words. For example the sentence such as "The school goes to boy" is rejected by English syntactic analyzer.

3) Semantic Analysis:

Semantic analysis is a structure created by the syntactic analyzer which assigns meanings. This component transfers linear sequences of words into structures. It shows how the words are associated with each other. Semantics focuses only on the literal meaning of words, phrases, and sentences. This only draws the dictionary meaning or the real meaning from the given text. The structures assigned by the syntactic analyzer always have assigned meaning.

The text is checked for meaningfulness. It is done by mapping syntactic structure and objects in the task domain. E.g. "colorless green idea". This would be rejected by the Symantec analysis as colorless here; green doesn't make any sense.

4) Discourse Integration:

The meaning of any sentence depends upon the meaning of the sentence just before it. Furthermore, it also brings about the meaning of immediately succeeding sentence. For example, "He wanted that", in this sentence the word "that" depends upon the prior discourse context.

5) Pragmatic Analysis:

Pragmatic analysis concerned with the overall communicative and social content and its effect on interpretation. It means abstracting or deriving the meaningful use of language in situations. In this analysis, what was said is reinterpreted on what it truly meant. It contains deriving those aspects of language which necessitate real world knowledge.

E.g., "close the window?" should be interpreted as a request instead of an order.

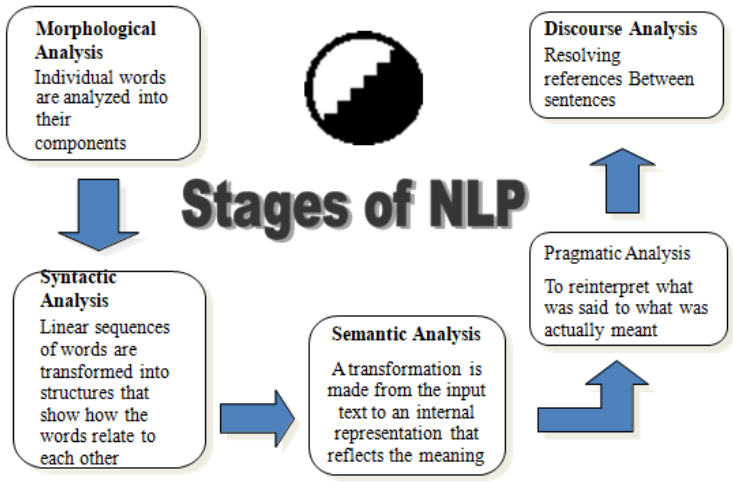


Figure 1.6.2 Stages of NLP

Morphological Analysis:

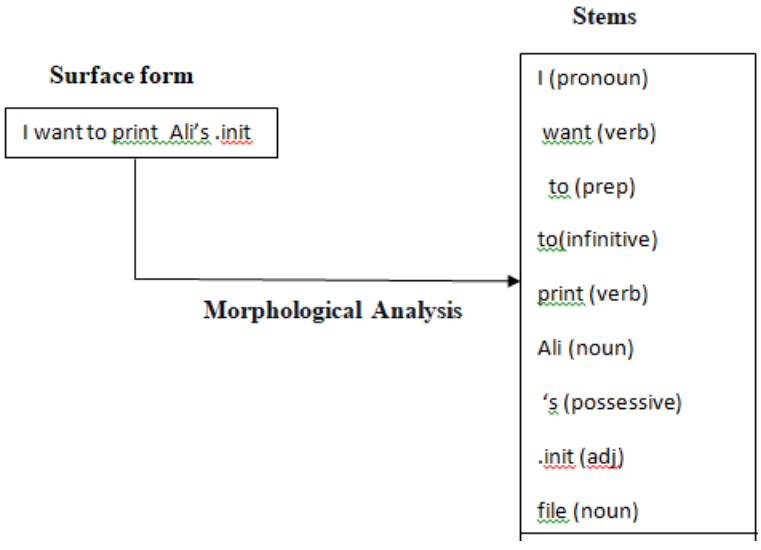
Consider we have an English interface to an operating system and the following sentence is typed: "I want to print Bill's .init file ".

Morphological analysis must do the following things:

• Pull apart the word "Bill's" into proper noun "Bill" and the possessive suffix "'s".

• Recognize the sequence ".init" as a file extension that is functioning as an adjective in the sentence.

This process will usually assign syntactic categories to all the words in the sentence. Consider the word "prints". This word is either a plural noun or a third person singular verb (he prints).



Syntactic analysis:

This method examines the structure of a sentence and performs detailed analysis of the sentence and semantics of the statement. In order to perform this, the system is expected to have through knowledge of the grammar of the language. The basic unit of any language is sentence, made up of group of words, having their own meanings and linked together to present an idea or thought. Apart from having meanings, words fall under categories called parts of speech. In English languages, there are eight different parts of speech. They are nouns, pronoun, adjectives, verb, adverb, prepositions, conjunction and interjections.

In English language, a sentence S is made up of a noun phrase (NP) and a verb phrase (VP), i.e.

S=NP+VP

The given noun phrase (NP) normally can have an article or delimiter (D) or an adjective (ADJ) and the noun (N), i.e.

NP=D+ADJ+N

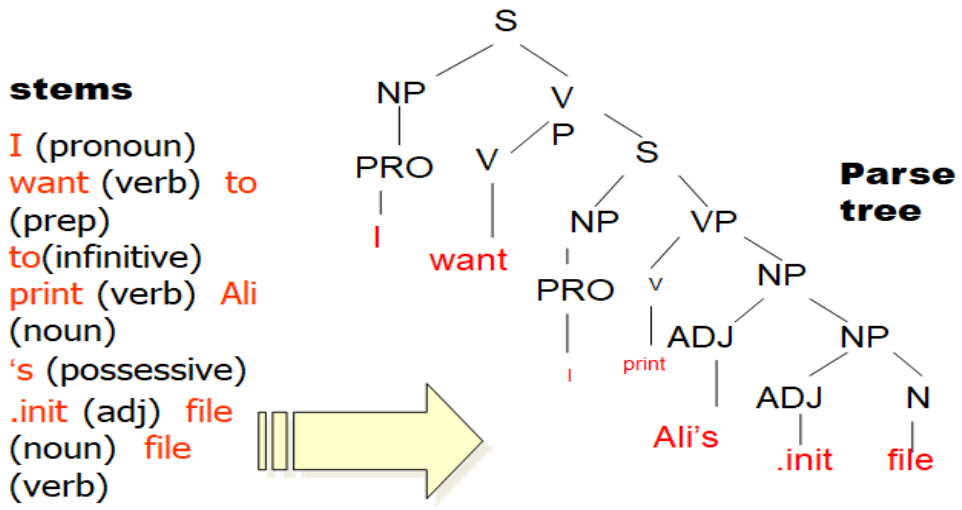
Also a noun phrase may have a prepositional phrase (PP) which has a preposition (P), a delimiter (D) and the noun (N), i.e.

PP=D+P+N

The verb phrase (VP) has a verb (V) and the object of the verb. The object of the verb may be a noun (N) and its determiner, i.e.

VP=V+N+D

These are some of the rules of the English grammar that helps one to construct a small parser for NLP.

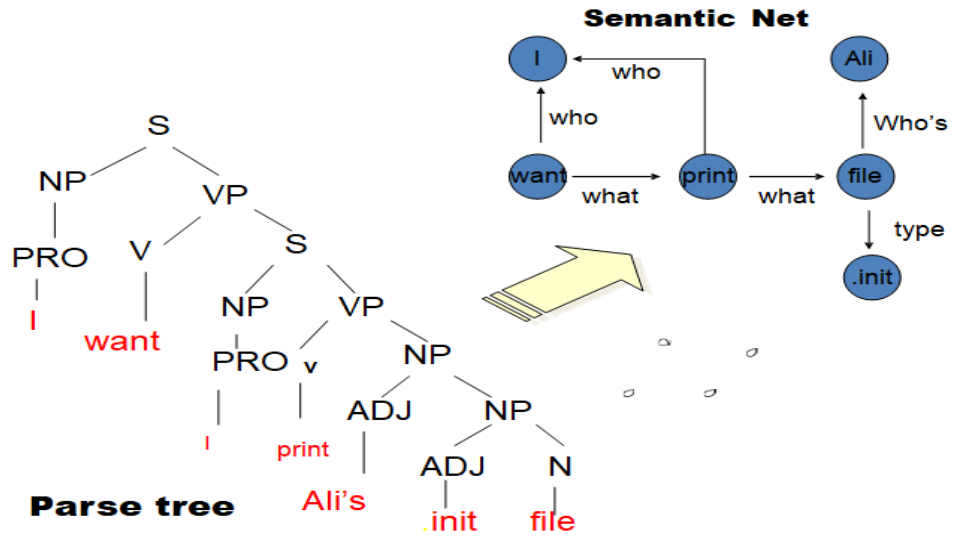


Syntactic analysis must exploit the results of morphological analysis to build a structural description of the sentence. The goal of this process, called parsing, is to convert the flat list of words that forms the sentence into a structure that defines the units that are represented by that flat list. The important thing here is that a flat sentence has been converted into a hierarchical structure and that the structures correspond to meaning units when semantic analysis is performed. Reference markers are shown in the parenthesis in the parse tree. Each one corresponds to some entity that has been mentioned in the sentence.

Semantic Analysis:

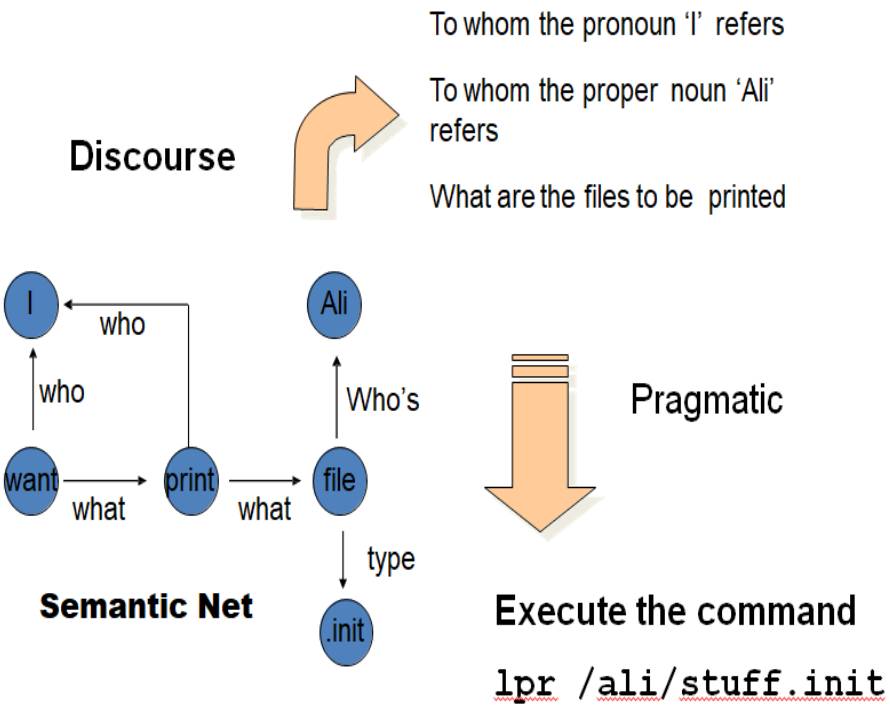
The structures created by the syntactic analyzer assigned meanings. Also, a mapping made between the syntactic structures and objects in the task domain. Moreover, Structures for which no such mapping possible may reject.

Semantic analysis must do two important things: It must map individual words into appropriate objects in the knowledge base or database It must create the correct structures to correspond to the way the meanings of the individual words combine with each other.



Discourse Integration:

The meaning of an individual sentence may depend on the sentences that precede it. And also, may influence the meanings of the sentences that follow it Specifically we do not know whom the pronoun "I" or the proper noun "Bill" refers to. To pin down these references requires an appeal to a model of the current discourse context, from which we can learn that the current user is USER068 and that the only person named "Bill" about whom we could be talking is USER073.Once the correct referent for Bill is known, we can also determine exactly which file is being referred to.



Pragmatic Analysis

Moreover, The structure representing what said reinterpreted to determine what was actually meant. The final step toward effective understanding is to decide what to do as a results. One possible thing to do is to record what was said as a fact and be done with it. For some sentences, whose intended effect is clearly declarative, that is precisely correct thing to do. But for other sentences, including this one, the intended effect is different. We can discover this intended effect by applying a set of rules that characterize cooperative dialogues. The final step in pragmatic processing is to translate, from the knowledge based representation to a command to be executed by the system.

Results of each of the main processes combine to form a natural language system.

The results of the understanding process are lpr /ali/stuff.init. All of the processes are important in a complete natural language understanding system. Not all programs are written with exactly these components. Sometimes two or more of them collapsed. Doing that usually results in a system that is easier to build for restricted subsets of English but one that is harder to extend to wider coverage.

1.10 Linguistic Background: An outline of English Syntax

- **Words**
- **The Elements of Simple Noun Phrases**
- **Verb Phrases and Simple Sentences**
- **Noun Phrases Revisited**
- **Adjective Phrases**
- **Adverbial Phrases**

1.Words

At first glance the most basic unit of linguistic structure appears to be the word. The word, though, is far from the fundamental element of study in linguistics; it is already the result of a complex set of more primitive parts. The study of **morphology** concerns the construction of words from more basic components corresponding roughly to meaning units. There are two basic ways that newwords are formed, traditionally classified as **inflectional** forms and **derivational** forms. Inflectional forms use a **root** form of a word and typically add a **suffix** so that the word appears in the appropriate form given the sentence. Verbs are the best examples of this in English. Each verb has a basic form that then is typically changed depending on the subject and the tense of the sentence. For example, the verb *sigh* will take suffixes such as *-s*, *-ing*, and *-ed* to create the verb forms *sighs*, *sighing*, and *sighed*, respectively. These new words are all verbs and share the same basic meaning. Derivational morphology involves the derivation of new words from other forms. The new words may be in completely different cate- gories from their subparts. For example, the noun *friend* is made into the adjec- tive *friendly* by adding the suffix *-ly*. A more complex derivation would allow you to derive the noun *friendliness* from the adjective form. There are many interesting issues concerned with how words are derived and how the choice of word form is affected by the syntactic structure of the sentence that constrains it.

Traditionally, linguists classify words into different categories based on their uses. Two related areas of evidence are used to divide words into categories. The first area concerns the word's contribution to the meaning of the phrase that contains it, and the second area concerns the actual syntactic structures

in which the word may play a role. For example, you might posit the class **noun** as those words that can be used to identify the basic type of object, concept, or place being discussed, and **adjective** as those words that further qualify the object, concept, or place. Thus *green* would be an adjective and *book* a noun, as shown in the phrases *the green book* and *green books*. But things are not so simple: *green* might play the role of a noun, as in *That green is lighter than the other,* and *book* might play the role of a modifier, as in *the book worm*. In fact, most nouns seem to be able to be used as a modifier in some situations. Perhaps the classes should be combined, since they overlap a great deal. But other forms of evidence exist. Consider what words could complete the sentence *It's so . . . .*
You might say *It's so green, It's so hot, It's so true,* and so on. Note that although *book* can be a modifier in *the book worm,* you cannot say **It's so book* about anything. Thus there are two classes of modifiers: adjective modifiers and noun modifiers.

Consider again the case where adjectives can be used as nouns, as in *the green*. Not all adjectives can be used in such a way. For example, the noun phrase *the hot* can be used, given a context where there are hot and cold plates, in a sentence such as *The hot are on the table.* But this refers to the hot plates; it cannot refer to hotness in the way the phrase *the green* refers to green. With this evidence you could subdivide adjectives into two subclasses—those that can also be used to describe a concept or quality directly, and those that cannot. Alter- natively, however, you can simply say that *green* is ambiguous between being an adjective or a noun and, therefore, falls in both classes. Since *green* can behave like any other noun, the second solution seems the most direct.

Using similar arguments, we can identify four main classes of words in English that contribute to the meaning of sentences. These classes are nouns, adjectives, verbs, and adverbs. Sentences are built out of phrases centered on these four word classes. Of course, there are many other classes of words that are necessary to form sentences, such as articles, pronouns, prepositions, particles, quantifiers, conjunctions, and so on. But these classes are fixed in the sense that new words in these classes are rarely introduced into the language. New nouns, verbs, adjectives and adverbs, on the other hand, are regularly introduced into the language as it evolves. As a result, these classes are called the **open class words**, and the others are called the **closed class words**.

A word in any of the four open classes may be used to form the basis of a phrase. This word is called the **head** of the phrase and indicates the type of thing, activity, or quality that the phrase describes. For example, with noun phrases, the head word indicates the general classes of objects being described. The phrases

the dog
the mangy dog
the mangy dog at the pound

are all noun phrases that describe an object in the class of dogs. The first describes a member from the class of all dogs, the second an object from the class of mangy dogs, and the third an object from the class of mangy dogs that are at the pound. The word *dog* is the head of each of these phrases.

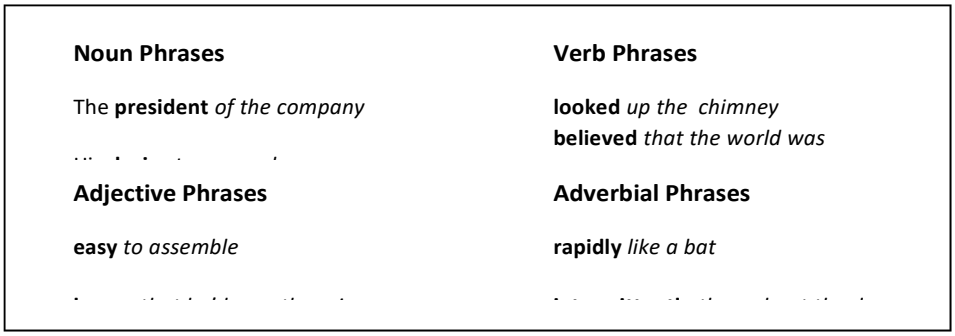


Figure 2.1 Examples of heads and complements

Similarly, the adjective phrases

hungry
very hungry hungry as a horse

all describe the quality of hunger. In each case the word *hungry* is the head.

In some cases a phrase may consist of a single head. For example, the word *sand* can be a noun phrase, *hungry* can be an adjective phrase, and *walked* can be a verb phrase. In many other cases the head requires additional phrases following it to express the desired meaning. For example, the verb *put* cannot form a verb phrase in isolation; thus the following words do not form a meaningful sentence:

*Jack put.

To be meaningful, the verb *put* must be followed by a noun phrase and a phrase describing a location, as in the verb phrase *put the dog in the house.* The phrase or set of phrases needed to complete the meaning of such a head is called the **complement** of the head. In the preceding phrase *put* is the head and *the dog in the house* is the complement. Heads of all the major classes may require comple- ments. Figure 2.1 gives some examples of phrases, with the head indicated by boldface and the complements by italics. In the remainder of this chapter, we will look at these different types of phrases in more detail and see how they are structured and how they contribute to the meaning of sentences.

2.1 The Elements of Simple Noun Phrases

Noun phrases (NPs) are used to refer to things: objects, places, concepts, events, qualities, and so on. The simplest NP consists of a single pronoun: *he, she, they, you, me, it, I,* and so on. Pronouns can refer to physical objects, as in the sentence

It hid under the rug.

to events, as in the sentence

Once I opened the door, I regretted *it* for months. and to qualities, as in the sentence

He was so angry, but he didn't show *it.*

Pronouns do not take any modifiers except in rare forms, as in the sentence He who hesitates is lost.

Another basic form of noun phrase consists of a **name** or **proper noun**, such as *John* or *Rochester.* These nouns appear in capitalized form in carefully written English. Names may also consist of multiple words, as in the *New York Times* and *Stratford-on-Avon.*

Excluding pronouns and proper names, the head of a noun phrase is usually a common noun. Nouns divide into two main classes:

count nouns—nouns that describe specific objects or sets of objects.
mass nouns—nouns that describe composites or substances.

Count nouns acquired their name because they can be counted. There may be one *dog* or many *dogs,* one *book* or several *books,* one *crowd* or several *crowds.* If a single count noun is used to describe a whole class of objects, it must be in its plural form. Thus you can say *Dogs are friendly* but not **Dog is friendly.*

Mass nouns cannot be counted. There may be *some water, some wheat,* or *some sand.* If you try to count with a mass noun, you change the meaning. For example, *some wheat* refers to a portion of some quantity of wheat, whereas *one wheat* is a single type of wheat rather than a single grain of wheat. A mass noun can be used to describe a whole class of material without using a plural form. Thus you say *Water is necessary for life,* not **Waters are necessary for life.*

In addition to a head, a noun phrase may contain **specifiers** and **qualifiers** preceding the head. The qualifiers further describe the general class of objects identified by the head, while the specifiers indicate how many such objects are being described, as well as how the objects being described relate to the speaker and hearer. Specifiers are constructed out of **ordinals** (such as *first* and *second*), **cardinals** (such as *one* and *two*), and **determiners**. Determiners can be sub- divided into the following general classes:

articles—the words *the, a,* and *an*.

demonstratives—words such as *this, that, these,* and *those*.

possessives—noun phrases followed by the suffix *'s*, such as *John's* and *the fat man's,* as well as possessive pronouns, such as *her, my,* and *whose.*

wh-determiners—words used in questions, such as *which* and *what.*

quantifying determiners—words such as *some, every, most, no, any, both,* and *half.*

Number	First Person	Second Person	Third Person
singular	I	you	he (masculine) she (feminine) it (neuter)
plural	we	you	they

Figure 2.2 Pronoun system (as subject)

Number	First Person	Second Person	Third Person
singular	my	your	his, her, its
plural	our	your	their

Figure 2.3 Pronoun system (possessives)

A simple noun phrase may have at most one determiner, one ordinal, and one cardinal. It is possible to have all three, as in *the first three contestants.* An exception to this rule exists with a few quantifying determiners such as *many, few, several,* and *little.* These words can be preceded by an article, yielding noun phrases such as *the few songs we knew.* Using this evidence, you could subcategorize the quantifying determiners into those that allow this and those that don't, but the present coarse categorization is fine for our purposes at this time.

The qualifiers in a noun phrase occur after the specifiers (if any) and before the head. They consist of adjectives and nouns being used as modifiers. The following are more precise definitions:

adjectives—words that attribute qualities to objects yet do not refer to the qualities themselves (for example, *angry* is an adjective that attributes the quality of anger to something).

noun modifiers—mass or count nouns used to modify another noun, as in *the cook book* or *the ceiling paint can.*

Before moving on to other structures, consider the different inflectional forms that nouns take and how they are realized in English. Two forms of nouns—the singular and plural forms—have already been mentioned. Pronouns take forms based on **person** (first, second, and third) and **gender** (masculine, feminine, and neuter). Each of these distinctions reflects a systematic analysis that is almost wholly explicit in some languages, such as Latin, while implicit in others. In French, for example, nouns are classified by their gender. In English many of these distinctions are not explicitly marked except in a few cases. The pronouns provide the best example of this. They distinguish **number**, **person**, **gender,** and **case** (that is, whether they are used as possessive, subject, or object), as shown in Figures 2.2 through 2.4.

Number	First Person	Second Person	Third Person
			him her it
singular	me	you	
plural	us	you	them

Mood	Example
declarative (or assertion)	The cat is sleeping.
yes/no question	Is the cat sleeping?
	What is sleeping? or Which cat is sleeping?

Figure 2.4 Pronoun system (as object)

Figure 2.5 Basic moods of sentences

Form	Examples	Example Uses
base	hit, cry, go, be	*Hit* the ball! I want to *go.*
simple present	hit, cries, go, am	The dog *cries* every day. I *am* thirsty.
simple past	hit, cried, went, was	I *was* thirsty. I *went* to the store.
present participle	hitting, crying, going, being	I'm *going* to the store. *Being* the last in line aggravates me.
past participle	hit, cried, gone, been	I've *been* there before. The cake was *gone.*

Figure 2.6 The five verb forms

2.2 Verb Phrases and Simple Sentences

While an NP is used to refer to things, a sentence (S) is used to assert, query, or command. You may assert that some sentence is true, ask whether a sentence is true, or command someone to do something described in the sentence. The way a sentence is used is called its **mood**. Figure 2.5 shows four basic sentence moods.

A simple declarative sentence consists of an NP, the subject, followed by a verb phrase (VP), the predicate. A simple VP may consist of some adverbial

Tense	The Verb Sequence	Example
simple present	simple present	He walks to the store.
simple past	simple past	He walked to the store.
simple future	*will* + infinitive	He will walk to the store.
present perfect	*have* in present + past participle	He has walked to the store.
future perfect	*will* + *have* in infinitive + past participle	I will have walked to the store.
past perfect (or pluperfect)	*have* in past + past participle	I had walked to the store.

Figure 2.7 The basic tenses

Tense	Structure	Example
present	*be* in present	He is walking.
past	+ present participle	He was walking.
future	*be* in past	He will be walking.
present perfect progressive	*have* in present + *be* in past participle + present participle	He has been walking.
future perfect progressive	*will* + *have* in present + *be* as past participle + present participle	He will have been walking.
past perfect progressive	*have* in past + *be* in past participle + present participle	He had been walking.

Figure 2.8 The progressive tenses

modifiers followed by the head verb and its complements. Every verb must appear in one of the five possible forms shown in Figure 2.6.

Verbs can be divided into several different classes: the **auxiliary verbs**, such as *be, do,* and *have;* the **modal verbs**, such as *will, can,* and *could;* and the **main verbs**, such as *eat, ran,* and *believe.* The auxiliary and modal verbs usually take a verb phrase as a complement, which produces a sequence of verbs, each the head of its own verb phrase. These sequences are used to form sentences with different tenses.

The **tense system** identifies when the proposition described in the sentence is said to be true. The tense system is complex; only the basic forms are outlined in Figure 2.7. In addition, verbs may be in the **progressive tense.** Corresponding

	First	Second	Third
Singular	I *am*	you *are*	he *is*
	I *walk*	you *walk*	she *walks*
Plural	we *are*	you *are*	they *are*
	we *walk*	you *walk*	they *walk*

Figure 2.9 Person/number forms of verbs

to the tenses listed in Figure 2.7 are the progressive tenses shown in Figure 2.8. Each progressive tense is formed by the normal tense construction of the verb *be* followed by a present participle.

Verb groups also encode person and number information in the first wordin the verb group. The person and number must agree with the noun phrase thatis the subject of the verb phrase. Some verbs distinguish nearly all the possibilities, but most verbs distinguish only the third person singular (by adding an *-s* suffix). Some examples are shown in Figure 2.9.

Transitivity and Passives

The last verb in a verb sequence is called the **main verb**, and is drawn from the open class of verbs. Depending on the verb, a wide variety of complement struc- tures are allowed. For example, certain verbs may stand alone with no comple- ment. These are called **intransitive** verbs and include examples such as *laugh* (for example, *Jack laughed*) and *run* (for example, *He will have been running*). Another common complement form requires a noun phrase to follow the verb. These are called **transitive** verbs and include verbs such as *find* (for example, *Jack found a key*). Notice that *find* cannot be intransitive (for example, **Jack found* is not a reasonable sentence), whereas *laugh* cannot be transitive (for example, **Jack laughed a key* is not a reasonable sentence). A verb like *run,* on the other hand, can be transitive or intransitive, but the meaning of the verb is different in each case (for example, *Jack ran* vs. *Jack ran the machine*).

Transitive verbs allow another form of verb group called the **passive** form, which is constructed using a *be* auxiliary followed by the past participle. In the passive form the noun phrase that would usually be in the object position is used in the subject position, as can be seen by the examples in Figure 2.10. Note that tense is still carried by the initial verb in the verb group. Also, even though the first noun phrase semantically seems to be the object of the verb in passive sentences, it is syntactically the subject. This can be seen by checking the pronoun forms. For example, *I was hit* is correct, not **Me was hit.* Furthermore, the tense and number agreement is between the verb and the syntactic subject. Thus you say *I was hit by them,* not **I were hit by them.*

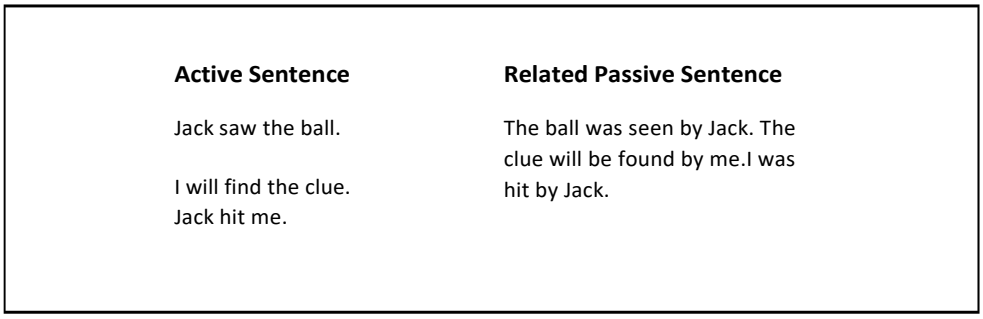


Figure 2.10 Active sentences with corresponding passive sentences

Some verbs allow two noun phrases to follow them in a sentence; for example, *Jack gave Sue a book* or *Jack found me a key.* In such sentences the second NP corresponds to the object NP outlined earlier and is sometimes called the **direct object**. The other NP is called the **indirect object**. Generally, such sentences have an equivalent sentence where the indirect object appears with a preposition, as in *Jack gave a book to Sue* or *Jack found a key for me.*

Particles

Some verb forms are constructed from a verb and an additional word called a **particle**. Particles generally overlap with the class of prepositions considered in the next section. Some examples are *up, out, over,* and *in.* With verbs such as *look, take,* or *put,* you can construct many different verbs by combining the verb with a particle (for example, *look up, look out, look over,* and so on). In some sentences the difference between a particle and a preposition results in two different readings for the same sentence. For example, *look over the paper* would mean reading the paper, if you consider *over* a particle (the verb is *look over*). In contrast, the same sentence would mean looking at something else behind or above the paper, if you consider *over* a preposition (the verb is *look*).

You can make a sharp distinction between particles and prepositions when the object of the verb is a pronoun. With a verb-particle sentence, the pronoun must precede the particle, as in *I looked it up.* With the prepositional reading, the pronoun follows the preposition, as in *I looked up it.* Particles also may follow the object NP. Thus you can say *I gave up the game to Mary* or *I gave the game up to Mary.* This is not allowed with prepositions; for example, you cannot say **Iclimbed the ladder up.*

Clausal Complements

Many verbs allow clauses as complements. Clauses share most of the same prop- erties of sentences and may have a subject, indicate tense, and occur in passivized forms. One common clause form consists of a sentence form preceded by the complementizer *that,* as in *that Jack ate the pizza.* This clause will be identified by the expression S[that], indicating a special subclass of S structures. This clause may appear as the complement of the verb *know,* as in *Sam knows*

that Jack ate the pizza. The passive is possible, as in *Sam knows that the pizza was eaten by Jack.*

Another clause type involves the infinitive form of the verb. The VP[inf] clause is simply a VP starting in the infinitive form, as in the complement of the verb *wish* in *Jack wishes to eat the pizza.* An infinitive sentence S[inf] form is also possible where the subject is indicated by a *for* phrase, as in *Jack wishes for Sam to eat the pizza.*

Another important class of clauses are sentences with complementizers that are wh-words, such as *who, what, where, why, whether,* and *how many.* These question clauses, S[WH], can be used as a complement of verbs such as *know,* as in *Sam knows whether we went to the party* and *The police know who committed the crime.*

Prepositional Phrase Complements

Many verbs require complements that involve a specific prepositional phrase (PP). The verb *give* takes a complement consisting of an NP and a PP with the preposition *to,* as in *Jack gave the book to the library.* No other preposition can be used. Consider

*Jack gave the book from the library. (OK only if *from the library*

modifies book.)

In contrast, a verb like *put* can take any PP that describes a location, as in Jack put the book in the box.
Jack put the book inside the box. Jack put the book by the door.

To account for this, we allow complement specifications that indicate prepo- sitional phrases with particular prepositions. Thus the verb *give* would have a complement of the form NP+PP[to]. Similarly the verb *decide* would have a complement form NP+PP[about], and the verb *blame* would have a complement form NP+PP[on], as in *Jack blamed the accident on the police.*

Verbs such as *put,* which take any phrase that can describe a location (complement NP+Location), are also common in English. While locations are typically prepositional phrases, they also can be noun phrases, such as *home,* or particles, such as *back* or *here.* A distinction can be made between phrases that describe locations and phrases that describe a path of motion, although many location phrases can be interpreted either way. The distinction can be made in some cases, though. For instance, prepositional phrases beginning with *to* generally indicate a path of motion. Thus they cannot be used with a verb such as *put* that requires a location (for example, **I put the ball to the box*). This distinction will be explored further in Chapter 4.

Figure 2.11 summarizes many of the verb complement structures found in English. A full list would contain over 40 different forms. Note that while the

Verb	Complement Structure	Example
laugh	Empty (intransitive)	Jack laughed.
find	NP (transitive)	Jack found a key.
give	NP+NP (bitransitive)	Jack gave Sue the paper.
give	NP+PP[to]	Jack gave the book to the library.
reside	Location phrase	Jack resides in Rochester
put	NP+Location phrase	Jack put the book inside.
speak	PP[with]+PP[about]	Jack spoke with Sue about the book.
try	VP[to]	Jack tried to apologize.
tell	NP+VP[to]	Jack told the man to go.
wish	S[to]	Jack wished for the man to go.
keep	VP[ing]	Jack keeps hoping for the best.
catch	NP+VP[ing]	Jack caught Sam looking in his desk.
watch	NP+VP[base]	Jack watched Sam eat the pizza.
regret	S[that]	Jack regretted that he'd eaten the whole thing.
tell	NP+S[that]	Jack told Sue that he was sorry.
seem	ADJP	Jack seems unhappy in his new job.
think	NP+ADJP	Jack thinks Sue is happy in her job.
know	S[WH]	Jack knows where the money is.

Figure 2.11 Some common verb complement structures in English

examples typically use a different verb for each form, most verbs will allow several different complement structures.

2.3 Noun Phrases Revisited

Section 2.2 introduced simple noun phrases. This section considers more com- plex forms in which NPs contain sentences or verb phrases as subcomponents.

All the examples in Section 2.2 had heads that took the null complement. Many nouns, however, may take complements. Many of these fall into the class of complements that require a specific prepositional phrase. For example, the noun *love* has a complement form PP[of], as in *their love of France,* the noun *reliance* has the complement form PP[on], as in *his reliance on handouts,* and the noun *familiarity* has the complement form PP[with], as in *a familiarity with computers.*

Many nouns, such as *desire, reluctance,* and *research,* take an infinitive VP form as a complement, as in the noun phrases *his desire to release the guinea pig, a reluctance to open the case again,* and *the doctor's research to find a cure for cancer.* These nouns, in fact, can also take the S[inf] form, as in *my hope for John to open the case again.*

Noun phrases can also be built out of clauses, which were introduced in the last section as the complements for verbs. For example, a *that* clause (S[that]) can be used as the subject of a sentence, as in the sentence *That George had the*

ring was surprising. Infinitive forms of verb phrases (VP[inf]) and sentences (S[inf]) can also function as noun phrases, as in the sentences *To own a car would be delightful* and *For us to complete a project on time would be unprecedented.* In addition, the **gerundive** forms (VP[ing] and S[ing]) can also function as noun phrases, as in the sentences *Giving up the game was unfortunate* and *John's giving up the game caused a riot.*

Relative clauses involve sentence forms used as modifiers in noun phrases. These clauses are often introduced by **relative pronouns** such as *who, which, that,* and so on, as in

The man *who gave Bill the money* . . . The rug *that George gave to Ernest* . . .

The man *whom George gave the money to* . . .

In each of these relative clauses, the embedded sentence is the same structure as a regular sentence except that one noun phrase is missing. If this missing NP is filled in with the NP that the sentence modifies, the result is a complete sentence that captures the same meaning as what was conveyed by the relative clause. The missing NPs in the preceding three sentences occur in the subject position, in the object position, and as object to a preposition, respectively. Deleting the relative pronoun and filling in the missing NP in each produces the following:

The man gave Bill the money. George gave *the rug* to Ernest. George gave the money to *the man.*

As was true earlier, relative clauses can be modified in the same ways as regular sentences. In particular, passive forms of the preceding sentences would be as follows:

Bill was given the money by the man. The rug was given to Ernest by George.
The money was given to the man by George.

Correspondingly, these sentences could have relative clauses in the passive form as follows:

The man *Bill was given the money by* . . .

The rug *that was given to Ernest by George* . . .

The man whom *the money was given to by George* . . .

Notice that some relative clauses need not be introduced by a relative pronoun. Often the relative pronoun can be omitted, producing what is called a **base relative clause**, as in the NP *the man George gave the money to.* Yet another form deletes the relative pronoun and an auxiliary *be* form, creating a **reduced relative clause**, as in the NP *the man given the money,* which means the same as the NP *the man who was given the money.*

2.4 Adjective Phrases

You have already seen simple adjective phrases (ADJPs) consisting of a single adjective in several examples. More complex adjective phrases are also possible, as adjectives may take many of the same complement forms that occur with verbs. This includes specific prepositional phrases, as with the adjective *pleased*, which takes the complement form PP[with] (for example, *Jack was pleased with the prize*), or *angry* with the complement form PP[at] (for example, *Jack was angry at the committee*). *Angry* also may take an S[that] complement form, as in *Jack was angry that he was left behind.* Other adjectives take infinitive forms, such as the adjective *willing* with the complement form VP[inf], as in *Jack seemed willing to lead the chorus.*

These more complex adjective phrases are most commonly found as the complements of verbs such as *be* or *seem*, or following the head in a noun phrase. They generally cannot be used as modifiers preceding the heads of noun phrases (for example, consider **the angry at the committee man* vs. *the angry man* vs. *the man angry at the committee*).

Adjective phrases may also take a degree modifier preceding the head, as in the adjective phrase *very angry* or *somewhat fond of Mary*. More complex degree modifications are possible, as in *far too heavy* and *much more desperate.* Finally, certain constructs have degree modifiers that involve their own complement forms, as in *too stupid to come in out of the rain, so boring that everyone fell asleep,* and *as slow as a dead horse.*

2.5 Adverbial Phrases

You have already seen adverbs in use in several constructs, such as indicators of degree (for example, *very, rather, too*), and in location phrases (for example, *here, everywhere*). Other forms of adverbs indicate the manner in which something is done (for example, *slowly, hesitantly*), the time of something (for example, *now, yesterday*), or the frequency of something (for example, *frequently, rarely, never*).

Adverbs may occur in several different positions in sentences: in the sentence initial position (for example, ***Then**, Jack will open the drawer*), in the verb sequence (for example, *Jack **then** will open the drawer, Jack will **then** open the drawer*), and in the sentence final position (for example, *Jack opened the drawer **then***). The exact restrictions on what adverb can go where, however, is quite idiosyncratic to the particular adverb.

In addition to these adverbs, adverbial modifiers can be constructed out of a wide range of constructs, such as prepositional phrases indicating, among other things, location (for example, *in the box*) or manner (for example, *in great haste*); noun phrases indicating, among other things, frequency (for example, *every day*); or clauses indicating, among other things, the time (for example, *when the bomb exploded*). Such adverbial phrases, however, usually cannot occur except in the sentence initial or sentence final position. For example, we can say *Every day*

John opens his drawer or *John opens his drawer every day*, but not **John every day opens his drawer*.

Because of the wide range of forms, it generally is more useful to consider adverbial phrases (ADVPs) by function rather than syntactic form. Thus we can consider manner, temporal, duration, location, degree, and frequency adverbial phrases each as its own form. We considered the location and degree forms earlier, so here we will consider some of the others.

Temporal adverbials occur in a wide range of forms: adverbial particles (for example, *now*), noun phrases (for example, *today, yesterday*), prepositional phrases (for example, *at noon, during the fight*), and clauses (for example, *when the clock struck noon, before the fight started*).

Frequency adverbials also can occur in a wide range of forms: particles (for example, *often*), noun phrases (for example, *every day*), prepositional phrases (for example, *at every party*), and clauses (for example, *every time that John comes for a visit*).

Duration adverbials appear most commonly as prepositional phrases (for example, *for three hours, about 20 feet*) and clauses (for example, *until the moon turns blue*).

Manner adverbials occur in a wide range of forms, including particles (for example, *slowly*), noun phrases (for example, *this way*), prepositional phrases (for example, *in great haste*), and clauses (for example, *by holding the embers at the end of a stick*).

In the analyses that follow, adverbials will most commonly occur as modifiers of the action or state described in a sentence. As such, an issue arises as to how to distinguish verb complements from adverbials. One distinction is that adverbial phrases are always optional. Thus you should be able to delete the adverbial and still have a sentence with approximately the same meaning (missing, obviously, the contribution of the adverbial). Consider the sentences

Jack put the box by the door. Jack ate the pizza by the door.

In the first sentence the prepositional phrase is clearly a complement, since deleting it to produce **Jack put the box* results in a nonsensical utterance. On the other hand, deleting the phrase from the second sentence has only a minor effect: *Jack ate the pizza* is just a less general assertion about the same situation described by *Jack ate the pizza by the door*.

PBR VITS (AUTONOMOUS)

IV B.TECH CSE-IOT

NATURAL LANGUAGE PROCESSING

TEXT BOOK: James Allen, **Natural Language Understanding**, 2nd Edition, 2003, Pearson Education

Course Instructor: Dr KV Subbaiah, M.Tech, Ph.D, Professor, Dept. of CSE

UNIT–II Grammars and Parsing

Grammars and Parsing- Top- Down and Bottom-Up Parsers, Transition Network Grammars, Feature Systems and Augmented Grammars, Morphological Analysis and the Lexicon, Parsing with Features, Augmented Transition Networks, Bayes Rule, Shannon game, Entropy and Cross Entropy.

2.1 Grammars and Parsing:

Natural language has an underlying structure usually referred to under the heading of Syntax. The fundamental idea of syntax is that words group together to form so-called constituents i.e. groups of words or phrases which behave as a single unit. These constituents can combine together to form bigger constituents and eventually sentences.

The instance, *John*, *the man*, *the man with a hat* ***and*** *almost every man* are constituents (called Noun Phrases or NP for short) because they all can appear in the same syntactic context (they can all function as the subject or the object of a verb for instance). Moreover, the NP constituent the man with a hat can combine with the VP (Verb Phrase) constituent run to form a S (sentence) constituent.

Grammar: It is a set of rules which checks whether a string belongs to a particular language a not.

A program consists of various strings of characters. But, every string is not a proper or meaningful string. So, to identify valid strings in a language, some rules should be specified to check whether the string is valid or not. These rules are nothing but make **Grammar**.

Example − In English Language, Grammar checks whether the string of characters is acceptable or not, i.e., checks whether nouns, verbs, adverbs, etc. are in the proper sequence.

Grammar

Context-Free Grammar(CFG):

Context-free grammar (CFG) is a type of formal grammar used in natural language processing (NLP) and computational linguistics. It provides a way to describe the structure of a language in terms of a set of rules for generating its sentences.

Formally, Context-Free Grammar (G) can be defined as − It is a 4-tuple (V,T,P,S)

- V is a set of Non-Terminals or Variables. Examples N,NP, V, VP, PP, Det etc
- T is a set of terminals. ∑={names, things, places & verbs}
- P is a set of Productions or set of rules
- S is a starting symbol

The following Grammar productions are used in NLP parsers.

1. S -> NP VP
2. NP -> ART N
3. NP -> ART ADJ N
4. VP -> V
5. VP -> V NP

Grammar rule	Example
S → NPVP	I + want a morning flight
NP → Pronoun	I
NP → Proper-Noun	Los Angeles
NP → Det Nominal	a flight
Nominal → Nominal Noun	morning flight
Nominal → Noun	flights
VP → Verb	do
VP → Verb NP	want + a flight
VP → Verb NP PP	leave + Boston + in the morning
VP → Verb PP	leaving + on Thursday
PP → Preposition NP	from + Los Angeles

Derivations:

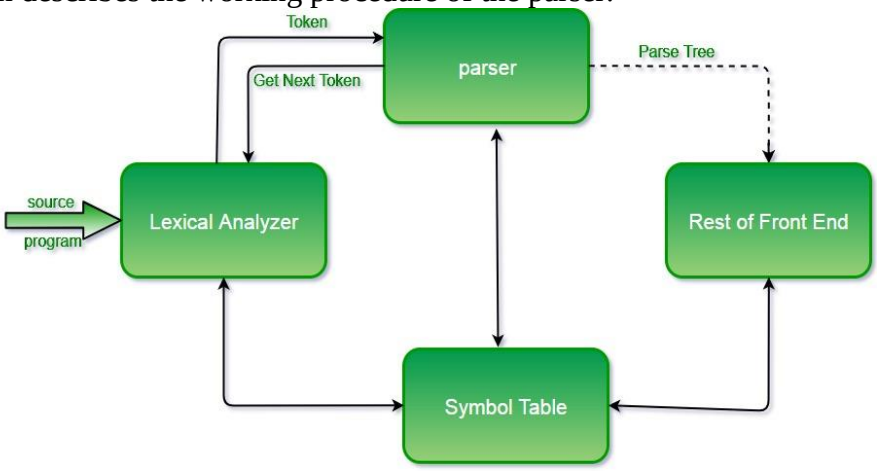
A derivation is a sequence of rule applications that derive a terminal string $w = w1 \ldots wn$ from S
• For example:

S→NP VP
Pro VP
I VP
I Verb NP
I prefer NP
I prefer Det Nom
I prefer a Nom
I prefer a Nom Noun
I prefer a Noun Noun
I prefer a morning Noun
I prefer a morning flight

Parsing:

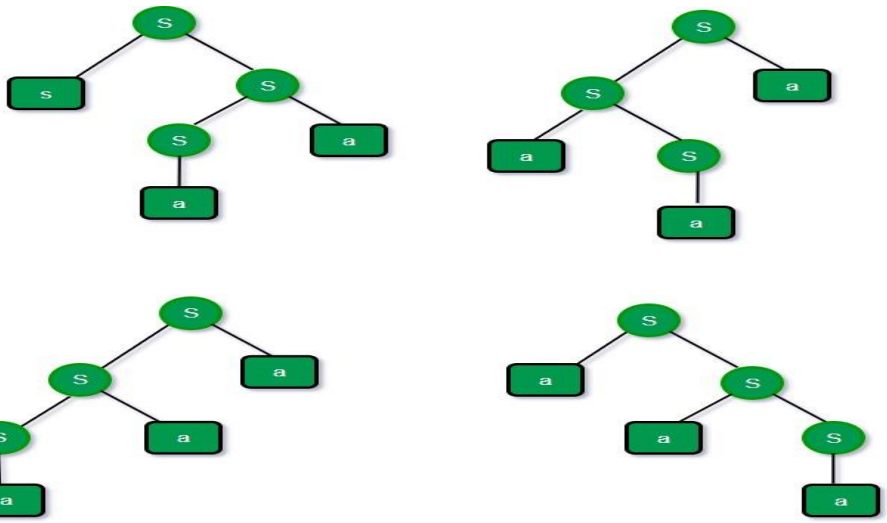
In the syntax analysis phase, a compiler verifies whether or not the tokens generated by the lexical analyzer are grouped according to the syntactic rules of the language. This is done by a parser.
The parser obtains a string of tokens from the lexical analyzer and verifies that the string can be the grammar for the source language. It detects and reports any syntax errors and produces a parse tree from which intermediate code can be generated.

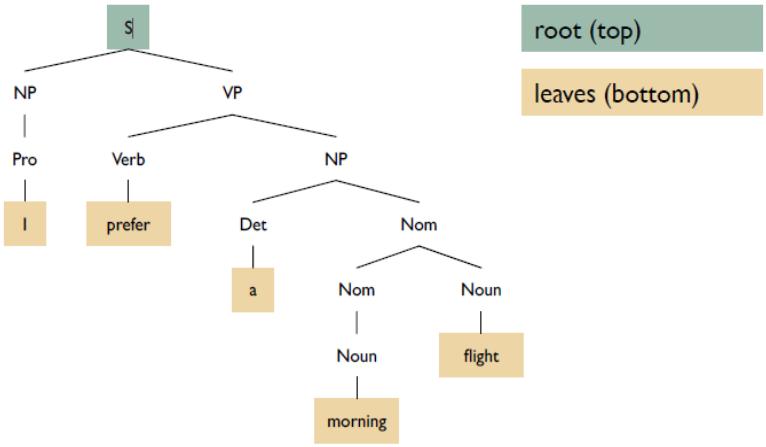
The following diagram describes the working procedure of the parser.



Ambiguity
A grammar that produces more than one parse tree for some sentence is said to be ambiguous.
Eg- consider a grammar
S -> aS | Sa | a

Now for string aaa, we will have 4 parse trees, hence ambiguous





Basic concepts of parsing:

Two problems for grammar *G* and string *w*:

- Recognition: determine if *G* accepts *w*
- Parsing: retrieve (all or some) parse trees assigned to *w* by *G*

Two basic search strategies:

- Top-down parser: start at the root of the tree
- Bottom-up parser: start at the leaves

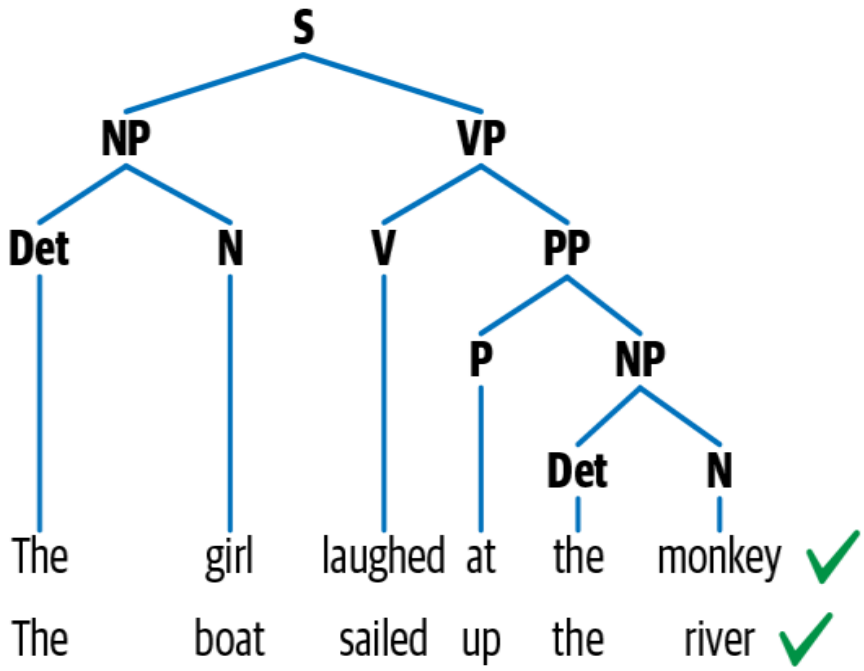


Figure 1-6. Syntactic structure of two syntactically similar sentences

2.2 Top-Down Parser

A parsing algorithm can be described as a procedure that searches through various ways of combining grammatical rules to find a combination that generates a tree that could be the structure of the input sentence. In other words, the algorithm will say whether a certain sentence is accepted by the grammar or not. The top-down parsing method is related in many artificial intelligence (AI) search applications.

A **top-down parser** starts with the S symbol and attempts to rewrite it into a sequence of terminal symbols that matches the classes of the words in the input sentence. The state of the parse at any given time can be represented as a list of symbols that are the result of operations applied so far, called the symbol list.

For example, the parser starts in the state (5) and after applying the rule S -> NP VP the symbol list will be (NP VP). If it then applies the rule NP ->ART N, the symbol list will be (ART N VP), and so on.

1. S -> NP VP
2. NP -> ART N
3. NP -> ART ADJ N
4. VP -> V
5. VP -> V NP

The parser could continue in this fashion until the state consisted entirely of terminal symbols, and then it could check the input sentence to see if it matched. The lexical analyser will produce list of words from the given sentence. A very small lexicon for use in the examples is

cried: V
dogs: N, V
the: ART

Positions fall between the words, with 1 being the position before the first word. For example, here is a sentence with its positions indicated:

$_1$ The $_2$ dogs $_3$ cried $_4$

A typical parse state would be

((N VP) 2)

indicating that the parser needs to find an N followed by a VP, starting at position two. New states are generated from old states depending on whether the first symbol is a lexical symbol or not. If it is a lexical symbol, like N in the preceding example, and if the next word can belong to that lexical category, then you can update the state by removing the first symbol and updating the position counter. In this case, since the word *dogs* is listed as an N in the lexicon, the next parser state would be

((VP) 3)

which means it needs to find a VP starting at position 3. If the first symbol is a nonterminal, like VP, then it is rewritten using a rule from the grammar. For example, using rule 4 in the above Grammar, the new state would be

((V) 3)

which means it needs to find a V starting at position 3. On the other hand, using rule *5*, the new state would be

((V NP) 3)

A parsing algorithm that is guaranteed to find a parse if there is one must systematically explore every possible new state. One simple technique for this is called backtracking. Using this approach, rather than generating a single new state from the state ((VP) 3), you generate all possible new states. One of these is picked to be the next state and the rest are saved as backup states. If you ever reach a situation where the current state cannot lead to a solution, you simply pick a new current state from the list of backup states. Here is the algorithm in a little more detail.

2.3 A Simple Top-Down Parsing Algorithm

The algorithm manipulates a list of possible states, called the possibilities list. The first element of this list is the current state, which consists of a symbol list - and a word position In the sentence, and the remaining elements of the search state are the backup states, each indicating an alternate symbol-list—word-position pair. For example, the possibilities list

(((N) 2) ((NAME) 1) ((ADJ N) 1))

indicates that the current state consists of the symbol list (N) at position 2, and that there are two possible backup states: one consisting of the symbol list (NAME) at position 1 and the other consisting of the symbol list (ADJ N) at position 1.

Step	Current State	Backup States	Comment
1	**((S) 1)**		initial position
2	**((NP VP) 1)**		rewriting S by rule I
3	**((ART N VP) 1)**		rewriting NP by rules 2 &3
		((ART ADJ N VP) 1)	
4	**((N VP) 2)**		matching ART with *the*
		((ART ADJ N VP) 1)	
5	**((VP) 3)**		matching N with *dogs*
		((ART ADJ N VP) 1)	
6	**((V) 3)**		rewriting VP by rules 5—8
		((V NP) 3)	
		((ART ADJ N VP) 1)	
7			the parse succeeds as V is
			matched to *cried*, leaving
			an empty grammatical symbol
			list with an empty sentence

Figure 3.5 Top-down depth-first parse of 1 The 2 dogs 3 cried 4

The algorithm starts with the initial state ((S) 1) and no backup states.

1. Select the current state: Take the first state off the possibilities list and call it C. If the possibilities list is empty, then the algorithm fails (that is, no successful parse is possible).

2. If C consists of an empty symbol list and the word position is at the end of the sentence, then the algorithm succeeds.

3. Otherwise, generate the next possible states.
   3.1. If the first symbol on the symbol list of C is a lexical symbol, and the next word in the sentence can be in that class, then create a new state by removing the first symbol from the symbol list and updating the word position, and add it to the possibilities list.
   3.2. Otherwise, if the first symbol on the symbol list of C is a non-terminal, generate a new state for each rule in the grammar that can rewrite that nonterminal symbol and add them all to the possibilities list.

Consider an example. Using Grammar 3.4, Figure 3.5 shows a trace of the algorithm on the sentence *The dogs cried.* First, the initial S symbol is rewritten using rule 1 to produce a new current state of ((NP VP) 1) in step 2. The NP is then rewritten in turn, but since there are two possible rules for NP in the grammar, two possible states are generated: The new current state involves (ART N VP) at position 1, whereas the backup state involves (ART ADJ N VP) at position 1. In step 4 a word in category ART is found at position 1 of the sentence, and the new current state becomes (N VP). The backup state generated in step 3 remains untouched. The
parse continues in this fashion to step *5,* where two different rules can rewrite VP. The first rule generates the new current state, while the other rule is pushed onto the stack of backup states. The parse completes successfully in step 7, since the current state is empty and all the words in the input sentence have been accounted for.

Consider the same algorithm and grammar operating on the sentence
1 The 2 old 3 man 4 cried 5
In this case assume that the word *old* is ambiguous between an ADJ and an N and that the word *man* is ambiguous between an N and a V (as in the sentence *The sailors man the boats).* Specifically, the lexicon is
the: ART
old: ADJ, N
man: N, V
cried: V

The parse proceeds as follows (see Figure 3.6). The initial S symbol is rewritten by rule 1 to produce the new current state of ((NP VP) 1). The NP is rewritten in turn, giving the new state of ((ART N VP) 1) with a backup state of ((ART ADJ N VP) 1). The parse continues, finding *the* as an ART to produce the state ((N VP) 2) and then *old* as an N to obtain the state ((VP) 3). There are now two ways to rewrite the VP, giving us a current state of ((V) 3) and the backup states of ((V NP) 3) and ((ART ADJ N) 1) from before. The word *man* can be parsed as a V. giving the state (04). Unfortunately, while the symbol list is empty, the word position is not at the end of the sentence, so no new state can be generated and a backup state must be used. In the next cycle, step 8, ((V NP) 3) is attempted. Again *man* is taken as a V and the new state ((NP) 4) generated. None of the rewrites of NP yield a successful parse. Finally, in step 12, the last backup state, ((ART ADJ N VP) 1), is tried and leads to a successful parse.

Parsing as a Search Procedure

You can think of parsing as a special case of a search problem as defined in Al. In particular, the top-down parser in this section was described in terms of the following generalized search procedure. The possibilities list is initially set to the start state of the parse. Then you repeat the following steps until you have success or failure:

1. Select the first state from the possibilities list (and remove it from the list).
2. Generate the new states by trying every possible option from the selected state (there may be none if we are on a bad path).
3. Add the states generated in step 2 to the possibilities list.

Step	Current State	Backup States	Comment
1.	((S) 1)		
2.	((NP VP) 1)		S rewritten to NP VP
3.	((ART N VP) 1)	((ART ADJ N VP) 1)	NP rewritten producing two new states
4.	((N VP) 2)	((ART ADJ N VP) 1)	
5.	((VP) 3)	((ART ADJ N VP) 1)	the backup state remains
6.	((V) 3)	((V NP) 3) ((ART ADJ N VP) 1)	
7.	(( ) 4)	((V NP) 3) ((ART ADJ N VP) 1)	
8.	((V NP) 3)	((ART ADJ N VP) 1)	the first backup is chosen
9.	((NP) 4)	((ART ADJ N VP) 1)	
10.	((ART N) 4)	((ART ADJ N) 4) ((ART ADJ N VP) 1)	looking for ART at 4 fails
11.	((ART ADJ N) 4)	((ART ADJ N VP) 1)	fails again
12.	((ART ADJ N VP) 1)		now exploring backup state saved in step 3
13.	((ADJ N VP) 2)		
14.	((N VP) 3)		
15.	((VP) 4)		
16.	((V) 4)	((V NP) 4)	
17.	(( ) 5)		success!

Figure 3.6 A top-down parse of $_1$ *The* $_2$ *old* $_3$ *man* $_4$ *cried* $_5$

Figure 3.6 A top-down parse of *1 The2 old man 4 cried5*

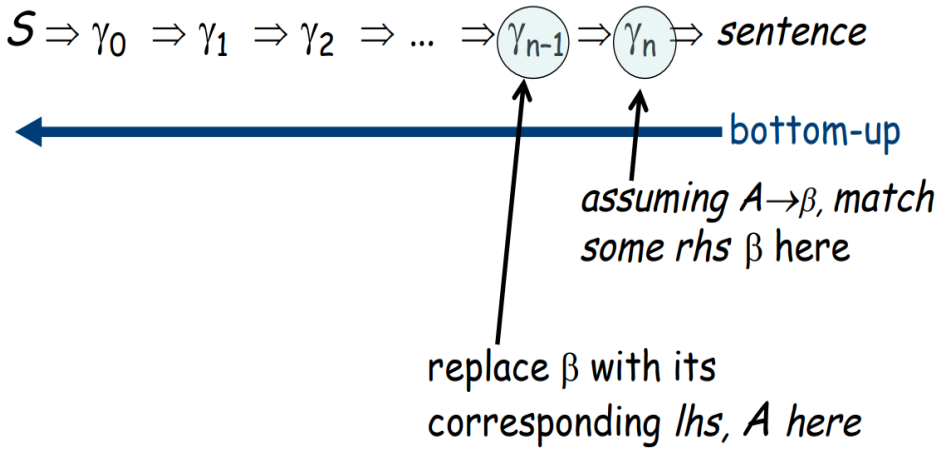
2.3 Bottom-Up Parser

A bottom-up parser builds derivation by working from input sentence back toward the start symbol S. The bottom-up parser is also known as **shift-reduce parser**.

The basic operation in bottom-up parsing is to take a sequence of symbols and match it to the right-hand side of the rules. You could build a bottom-up parser simply by formulating this matching process as a

search process. The state would simply consist of a symbol list, starting with the words in the sentence. Successor states could be generated by exploring all possible ways to :

- rewrite a word by its possible lexical categories
- replace a sequence of symbols that matches the right-hand side of a grammar rule by its left-hand side

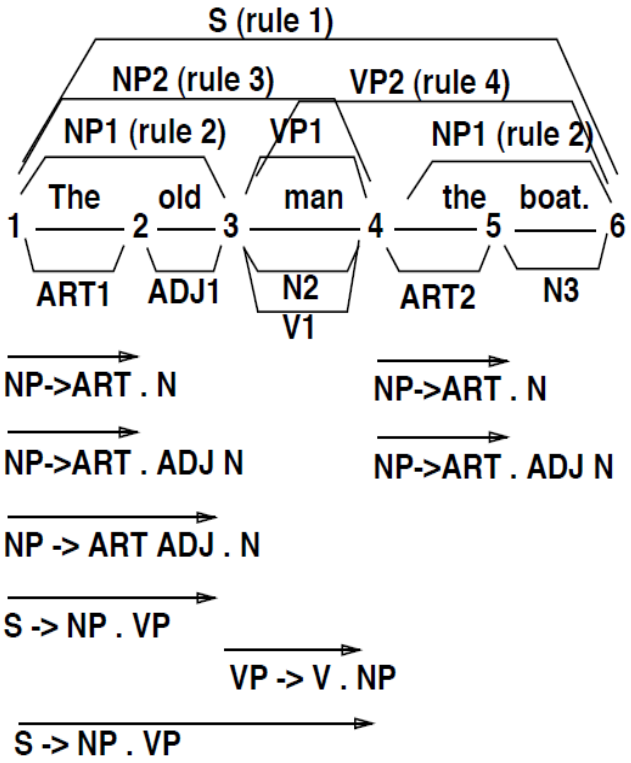


Bottom-up Chart Parsing Algorithm
Initialization: For every rule in the grammar of form S -> X1 ... Xk, add an arc labeled S -> o X1 ... Xk using the arc introduction algorithm.
Parsing: Do until there is no input left:
1. If the agenda is empty, look up the interpretations for the next word in the input and add them to the agenda.
2. Select a constituent from the agenda (let's call it constituent C from position p1 to p2).
3. For each rule in the grammar of form **X -> C X1 ... Xn**, add an active arc of form **X -> C o C o X1 .... Xn** from position p1 to p2.
4. Add C to the chart using the arc extension algorithm above.

Example1.

Grammar and Lexicon
Grammar:
1. S→NP VP
2. NP→ART N
3. NP→ART ADJ N

4. VP →V NP
Lexicon:
the: ART man: N, V
old: ADJ, N boat: N
Sentence: 1 The 2 old 3 man 4 the 5 boat 6



EXAMPLE 2: cats scratch people with claws

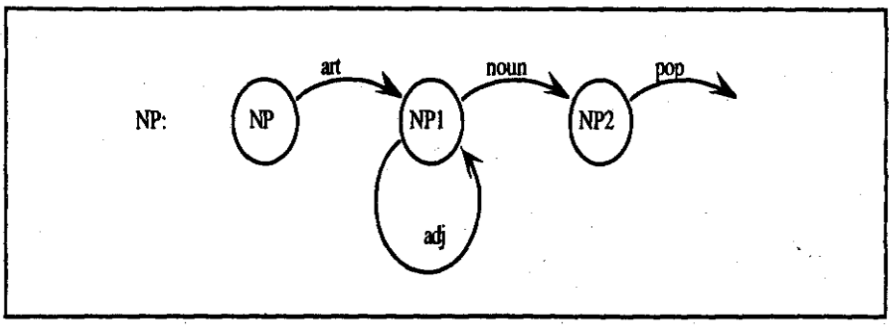
Shift-reduce parsing: one path

	cats scratch people with claws	
cats	*scratch people with claws*	SHIFT
N	*scratch people with claws*	REDUCE
NP	*scratch people with claws*	REDUCE
NP *scratch*	*people with claws*	SHIFT
NP V	*people with claws*	REDUCE
NP V *people*	*with claws*	SHIFT
NP V N	*with claws*	REDUCE
NP V NP	*with claws*	REDUCE
NP V NP *with*	*claws*	SHIFT
NP V NP P	*claws*	REDUCE
NP V NP P *claws*		SHIFT
NP V NP P N		REDUCE
NP V NP P NP		REDUCE
NP V NP PP		REDUCE
NP VP		REDUCE
S		REDUCE

What other search paths are there for parsing this sentence?

2.4 Transition Network Grammars

The Transition Network Grammars are based on nodes and edges labeled arcs. One of the nodes is specified as the initial state, or start state. Consider the network named NP in the following Grammar with the initial state labeled NP and each arc labelled with a word category.



Grammar 3.16

Starting at the initial state, you can traverse an arc if the current word in the sentence is in the category on the arc. If the arc is followed, the current word is updated to the next word. A phrase is a legal NP if there is a path from the node NP to a pop arc (an arc labeled pop) that accounts for every word in the phrase. This network recognizes the same set of sentences as the following context-free grammar:

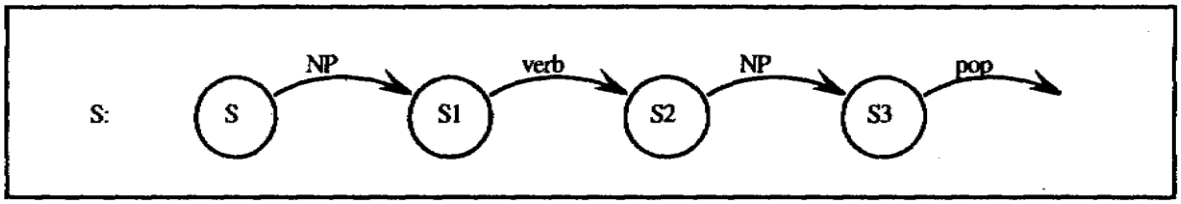
NP -> ART NP1

NP1 -> ADJ NP1

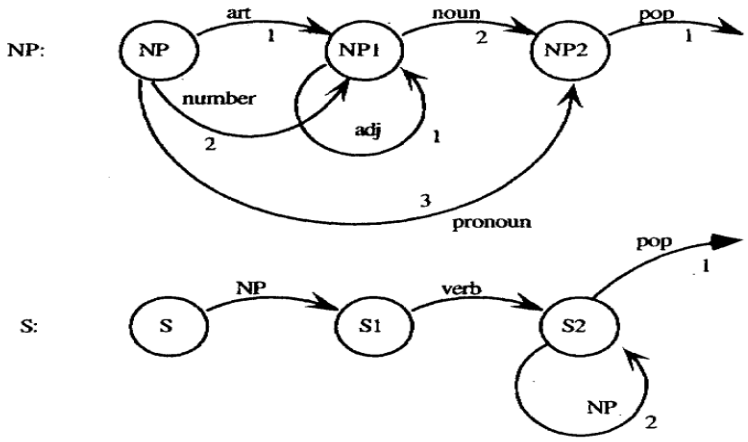
NP1 -> N

Consider parsing the NP *a purple cow* with this network. Starting at the node NP, you can follow the arc labelled art, since the current word is an article— namely, *a.* From node NP1 you can follow the arc labeled adj using the adjective *purple,* and finally, again from NP1, you can follow the arc labeled noun using the noun *cow.* Since you have reached a pop arc, *a purple cow* is a legal NP.

Consider finding a path through the S network for the sentence *The purple cow ate the grass.* Starting at node 5, to follow the arc labeled NP, you need to traverse the NP network. Starting at node NP, traverse the network as before for the input *the purple cow.* Following the pop arc in the NP network, return to the S network and traverse the arc to node S 1. From node S 1 you follow the arc labeled verb using the word *ate.* Finally, the arc labeled NP can be followed if you can traverse the NP network again. This time the remaining input consists of the words *the grass.* You follow the arc labeled art and then the arc labeled noun in the NP network; then take the pop arc from node NP2 and then another pop from node S3. Since you have traversed the network and used all the words in the sentence, *The purple cow ate the grass* is accepted as a legal sentence.



Grammar 3.17



Grammar 3.19

2.5 Feature Systems and Augmented Grammars

In natural languages there are often agreement restrictions between words and phrases. For example, the NP "*a men"* is not correct English because the article a indicates a single object while the noun "*men"* indicates a plural object; the noun phrase does not satisfy the number agreement restriction of English.

There are many other forms of agreement, including subject-verb agreement, gender agreement for pronouns, restrictions between the head of a phrase and the form of its complement, and so on. To handle such phenomena conveniently, the grammati cal formalism is extended to allow constituents to have features. For example, we might define a feature NUMBER that may take a value of either *s* (for singular) or p (for plural), and we then might write an augmented CFG rule such as

NP -> ART N only when NUMBER1 agrees with NUMBER2

This rule says that a legal noun phrase consists of an article followed by a noun, but only when the number feature of the first word agrees with the number feature of the second. This one rule is equivalent to two CFG rules that would use different terminal symbols for encoding singular and plural forms of all noun phrases, such as

NP-SING -> ART-SING N-SING
NP-PLURAL -> ART-PLURAL N-PLURAL

While the two approaches seem similar in ease-of-use in this one example, consider that all rules in the grammar that use an NP on the right-hand side would now need to be duplicated to include a rule for NP-SING and a rule for NP -PLURAL, effectively doubling the size of the grammar. And handling additional features, such as person agreement, would double the size of the grammar again and again. Using features, the size of the augmented grammar remains the same as the original one yet accounts for agreement constraints.

To accomplish this, a constituent is defined as a feature structure - a mapping from features to values that defines the relevant properties of the constituent. In the examples, feature names in formulas will be written in boldface. For example, a feature structure for a constituent ART1 that represents a particular use of the word a might be written as follows:

ART1: (CAT ART
ROOT a
NUMBER s**)**

This says it is a constituent in the category ART that has as its root the word a and is singular. Usually an abbreviation is used that gives the CAT value more prominence and provides an intuitive tie back to simple context-free grammars. In this abbreviated form, constituent ART1 would be written as

ART1: (ART **ROOT** a **NUMBER** s**)**

Feature structures can be used to represent larger constituents as well. To do this, feature structures themselves can occur as values. Special features based on the integers - 1, 2, 3, and so on - will stand for the first subconstituent, second subconstituent, and so on, as needed. With this, the representation of the NP constituent for the phrase "*a fish*" could be

NP1: (NP NUMBERs
1 (ART **ROOT** a
NUMBER s**)**
2 (N ROOT fish
NUMBER s**))**

Note that this can also be viewed as a representation of a parse tree shown in Figure 4.1, where the subconstituent features 1 and 2 correspond to the subconstituent links in the tree.

The rules in an augmented grammar are stated in terms of feature structures rather than simple categories. Variables are allowed as feature values so that a rule can apply to a wide range of situations. For example, a rule for simple noun phrases would be as follows:

(NP **NUMBER** ?n**) - (**ART **NUMBER** ?n**) (**N **NUMBER** ?n**)**

This says that an NP constituent can consist of two subconstituents, the first being an ART and the second being an N, in which the NUMBER feature in all three constituents is identical. According to this rule, constituent NP1 given previously is a legal constituent. On the other hand, the constituent is not allowed because the NUMBER feature of the N constituent is not identical to the other two NUMBER features**.**

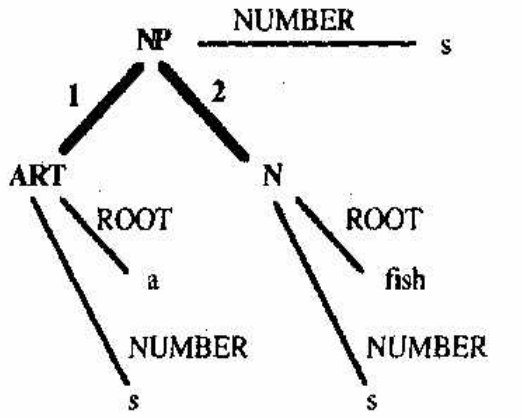


Figure 4.1 Viewing a feature structure as an extended parse tree

* (NP **1** (ART **NUMBER** s)
2 (N **NUMBER** s))
is not allowed by this rule because there is no NUMBER feature in the NP, and the constituent
*(NP **NUMBER** s
1 (ART **NUMBER** s)
2 (N **NUMBER** p))

BOX 4.1 Formalizing Feature Structures

There is an active area of research in the formal properties of feature structures. This work views a feature system as a formal logic. A feature structure is defined as a partial function from features to feature values. For example, the feature structure

ART1: (**CAT** ART
ROOT a
NUMBER *s*)

is treated as an abbreviation of the following statement in FOPC:

$$ART1(CAT) = ART \wedge ART1(ROOT) = a \wedge ART1(NUMBER) = s$$

Feature structures with disjunctive values map to disjunctions. The structure

THE1: (**CAT** ART
ROOT the
NUMBER {s p})

would be represented as

$$THE1(CAT) = ART \wedge THE1(ROOT) = the$$
$$\wedge (THE1(NUMBER) = s \vee THE1(NUMBER) = p)$$

Given this, agreement between feature values can be defined as equality equations.

2.6 Morphological Analysis and the Lexicon

Lexical Morphology: Lexical morphology is a specific type of morphology that refers to the **lexemes** in a language. A lexeme is a basic unit of lexical meaning; it can be a single word or a group of words.

Morphological Analysis: Morphological Analysis is the study of lexemes and how they are created. The discipline is particularly interested in **neologisms** (newly created words from existing words(root word)), **derivation**, and **compounding**. In morphological analysis each token will be analysed as follows:

token → lemma(root word) + part of speech + grammatical features

Examples: cats→ cat+N+plur

played →play+V+past

katternas →katt+N+plur+def+gen

Often non-deterministic (more than one solution):

plays →play+N+plur

plays → play+V+3sg

Derivation in Lexical Morphology:

Derivation refers to a way of creating new words by adding **affixes** to the root of a word - this is also known as **affixation**. There are two of affixes: **prefixes and suffixes**.

Prefixes: rewrite (root is write), unfair (root is fair)
Suffixes: wanted, wants, wanting (root is want)

Compounding: Compounding refers to the creation of new words by combining two or more existing words together. Now here are some examples of compounding:

Green + house = Greenhouse
Mother + in + law = Mother-in-law
Motor + bike = Motorbike
Cook + book = Cookbook
Foot + ball = Football

The lexicon must contain information about all the different words that can be used, including all the relevant feature value restrictions. When a word is ambiguous, it may be described by multiple entries in the lexicon, one for each different use.

Most English verbs, for example, use the same set of suffixes to indicate different forms: -s is added for third person singular present tense, -ed for past tense, -ing for the present participle, and so on.

The idea is to store the base form of the verb in the lexicon and use context-free rules to combine verbs with suffixes to derive the other entries. Consider the following rule for present tense verbs:

(V **ROOT** ?r **SUBCAT** ?s **VFORM** pres **AGR** 3s) ->
(V **ROOT** ?r **SUBCAT** ?s **VFORM** base) (+S)

where +S is a new lexical category that contains only the suffix morpheme -s. This rule, coupled with the lexicon entry

want:
(V **ROOT** want
SUBCAT {_np_vp:inf _np_vp:inf}
VFORM base)

would produce the following constituent given the input string want -s

want:
(V **ROOT** want
SUBCAT {_np_vp:inf _np_vp:inf}
VFORM pres **AGR** 3s)

Another rule would generate the constituents for the present tense form not in third person singular, which for most verbs is identical to the root form:

(V **ROOT** ?r **SUBCAT** ?s **VFORM** pres **AGR** {ls 2s lp 2p 3p})

—> (V **ROOT** ?r **SUBCAT** ?s **VFORM** base)

But this rule needs to be modified in order to avoid generating erroneous interpretations. Currently, it can transform any base form verb into a present tense form, which is clearly wrong for some irregular verbs. For instance, the base form be cannot be used as a present form (for example, *We be at the store). To cover these cases, a feature is introduced to identify irregular forms. Specifically, verbs with the binary feature +IRREGPRES have irregular present tense forms. Now the rule above can be stated correctly:

(V **ROOT** ?r **SUBCAT** ?s **VFORM** pres **AGR** (ls 2s lp 2p 3p))
—> (V **ROOT** ?r **SUBCAT** ?s **VFORM** base **IRREG-PRES** -)

The following Grammar 4.5 gives a set of rules for deriving different verb and noun forms using these features. Given a large set of features, the task of writing lexical entries appears very difficult. Most frameworks allow some mechanisms that help alleviate these problems.

Present Tense

1. (V **ROOT** ?r **SUBCAT** ?s **VFORM** pres **AGR** 3s) →
   (V **ROOT** ?r **SUBCAT** ?s **VFORM** base **IRREG-PRES** –) +S
2. (V **ROOT** ?r **SUBCAT** ?s **VFORM** pres **AGR** {1s 2s 1p 2p 3p}) →
   (V **ROOT** ?r **SUBCAT** ?s **VFORM** base **IRREG-PRES** –)

Past Tense

3. (V **ROOT** ?r **SUBCAT** ?s **VFORM** past **AGR** {1s 2s 3s 1p 2p 3p}) →
   (V **ROOT** ?r **SUBCAT** ?s **VFORM** base **IRREG-PAST** –) +ED

Past Participle

4. (V ROOT ?r SUBCAT ?s VFORM pastprt) →
   (V ROOT ?r SUBCAT ?s VFORM base **EN-PASTPRT** –) +ED
5. (V ROOT ?r SUBCAT ?s VFORM pastprt) →
   (V ROOT ?r SUBCAT ?s VFORM base **EN-PASTPRT** +) +EN

Present Participle

6. (V ROOT ?r SUBCAT ?s VFORM ing) →
   (V ROOT ?r SUBCAT ?s VFORM base) +ING

Plural Nouns

7. (N ROOT ?r AGR 3p) →
   (N ROOT ?r AGR 3s **IRREG-PL** –) +S

Grammar 4.5 Some lexical rules for common suffixes on verbs and nouns

Another commonly used technique is to allow the lexicon writing to define clusters of features, and then indicate a cluster with a single symbol rather than listing them all. Figure 4.6 contains a small lexicon. It contains many of the words to be used in the examples that follow. It contains three entries for the word "saw "- as a noun, as a regular verb, and as the irregular past tense form of the verb "see" - as illustrated in the sentences

The saw was broken.
Jack wanted me to saw the board in half.

I saw Jack eat the pizza.

With the lexicon in Figure 4.6 and Grammar 4.5, correct constituents for the following words can be derived: been, being, cries, cried, crying, dogs, saws (two interpretations), sawed, sawing, seen, seeing, seeds, wants, wanting, and wanted. For example, the word cries would be transformed into the sequence cry +s, and then rule 1 would produce the present tense entry from the base form in the lexicon.

```
a:      (CAT ART
         ROOT A1
         AGR 3s)
be:     (CAT V
         ROOT BE1
         VFORM base
         IRREG-PRES +
         IRREG-PAST +
         SUBCAT (_adjp _np))
cry:    (CAT V
         ROOT CRY1
         VFORM base
         SUBCAT _none)
dog:    (CAT N
         ROOT DOG1
         AGR 3s)
fish:   (CAT N
         ROOT FISH1
         AGR (3s 3p)
         IRREG-PL +)
happy:  (CAT ADJ
         SUBCAT _vp:inf)
he:     (CAT PRO
         ROOT HE1
         AGR 3s)
is:     (CAT V
         ROOT BE1
         VFORM pres
         SUBCAT (_adjp _np)
         AGR 3s)
Jack:   (CAT NAME
         AGR 3s)
man:    (CAT N1
         ROOT MAN1
         AGR 3s)
men:    (CAT N
         ROOT MAN1
         AGR 3p)
saw:    (CAT N
         ROOT SAW1
         AGR 3s)
saw:    (CAT V
         ROOT SAW2
         VFORM base
         SUBCAT _np)
saw:    (CAT V
         ROOT SEE1
         VFORM past
         SUBCAT _np)
see:    (CAT V
         ROOT SEE1
         VFORM base
         SUBCAT _np
         IRREG-PAST +
         EN-PASTPRT +)
seed:   (CAT N
         ROOT SEED1
         AGR 3s)
the:    (CAT ART
         ROOT THE1
         AGR (3s 3p))
to:     (CAT TO)
want:   (CAT V
         ROOT WANT1
         VFORM base
         SUBCAT (_np _vp:inf _np_vp:inf))
was:    (CAT V
         ROOT BE1
         VFORM past
         AGR (1s 3s)
         SUBCAT (_adjp _np))
were:   (CAT V
         ROOT BE
         VFORM past
         AGR (2s 1p 2p 3p)
         SUBCAT (_adjp _np))
```

Figure 4.6 A lexicon

2.7 Parsing with Features

The parsing algorithms are used for context-free grammars and also extended to handle augmented context-free grammars. This involves generalizing the algorithm for matching rules to constituents. For instance, the chart-parsing algorithms can be used an operation for extending active arcs with a new constituent. A constituent X could extend an arc of the form

C -> C1 ... Ci o X ... Cn

to produce a new arc of the form

C -> C1 ... Ci X o ... Cn

A similar operation can be used for grammars with features, but the parser may have to instantiate variables in the original arc before it can be extended by X. The key to defining this matching operation precisely is to remember the definition of grammar rules with features. A rule such as:

1. (NP **AGR** ?a) -> o (**ART AGR** ?a) (N **AGR** ?a)

says that an NP can be constructed out of an ART and an N if all three agree on the AGR feature. It does not place any restrictions on any other features that the NP, ART, or N may have. Thus, when matching constituents against this rule, the only thing that matters is the AGR feature. All other features in the constituent can be ignored. For instance, consider extending arc 1 with the constituent

2. (**ART ROOT** A **AGR** 3s)

To make arc 1 applicable, the variable ?a must be instantiated to 3s, producing

3. (NP **AGR** 3s) -> o (**ART AGR** 3s) (N **AGR** 3s)

This arc can now be extended because every feature in the rule is in constituent 2:

4. (NP **AGR** 3s) -> (**ART AGR** 3s) o (N **AGR** 3s)

Now, consider extending this arc with the constituent for the word dog:

5. (N **ROOT** DOG1 **AGR** 3s)

This can be done because the AGR features agree. This completes the arc

6. (NP **AGR** 3s) —> (**ART AGR** 3s) (N **AGR** 3s)

This means the parser has found a constituent of the form (NP AGR 3s).

This algorithm can be specified more precisely as follows: Given an arc A, where the constituent following the dot is called NEXT, and a new constituent X, which is being used to extend the arc,

(a.) Find an instantiation of the variables such that all the features specified in NEXT are found in X.

(b.) Create a new arc A', which is a copy of A except for the instantiations of the variables determined in step (a).

(c.) Update A' as usual in a chart parser.

For instance, let A be arc 1, and X be the ART constituent 2. Then NEXT will be (ART AGR ?a). In step a, NEXT is matched against X, and you find that ?a must be instantiated to 3s. In step b, a new copy of A is made, which is shown as arc 3. In step c, the arc is updated to produce the new arc shown as arc 4.

Figure 4.10 contains the final chart produced from parsing the sentence He wants to cry using Grammar 4.8 & Grammar 4.7.

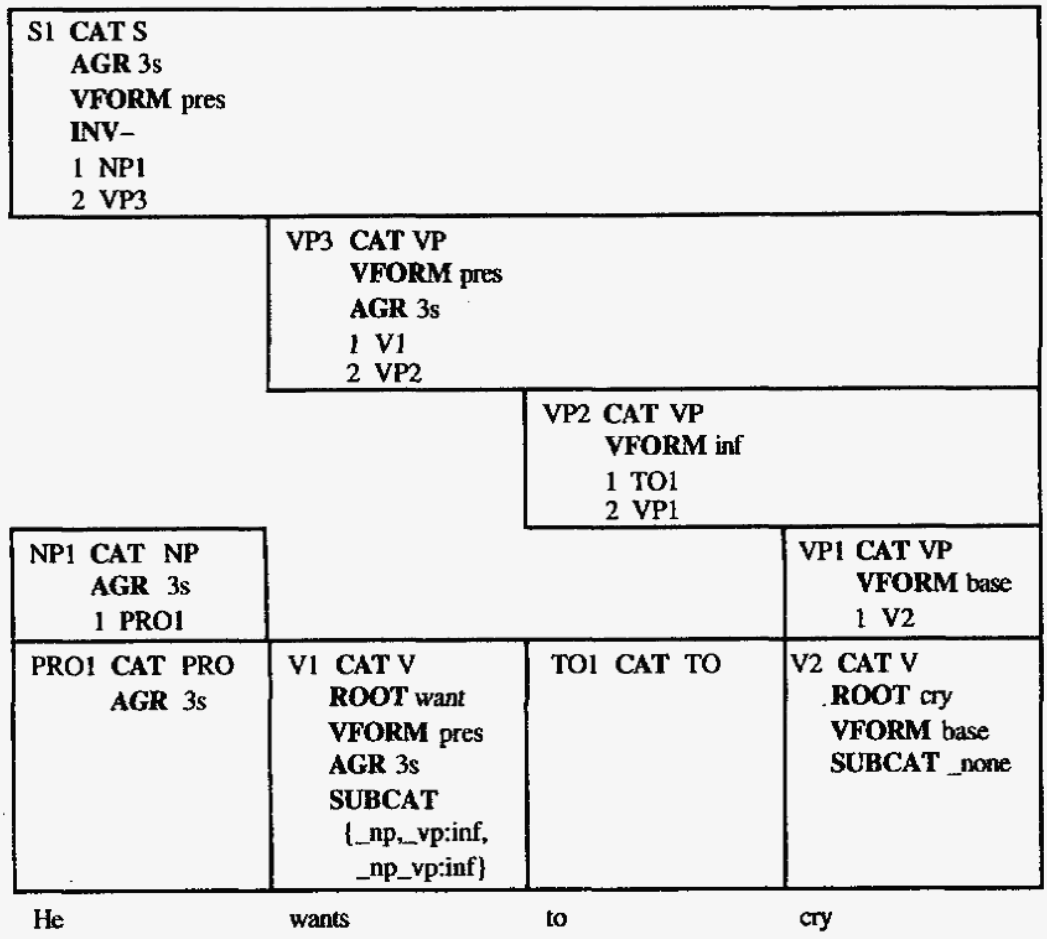


Figure 4.10 The chart for *He wants to cry.*

1. (S **INV** – **VFORM** ?v{pres past} **AGR** ?a) →
   (NP **AGR** ?a) (*VP VFORM ?v{pres past} AGR ?a*)
2. (NP **AGR** ?a) → (ART **AGR** ?a) (*N AGR ?a*)
3. (NP **AGR** ?a) → (*PRO AGR ?a*)
4. (VP **AGR** ?a **VFORM** ?v) → (*V SUBCAT _none AGR ?a VFORM ?v*)
5. (VP **AGR** ?a **VFORM** ?v) → (*V SUBCAT _np AGR ?a VFORM ?v*) NP
6. (VP **AGR** ?a **VFORM** ?v) →
   (*V SUBCAT _vp:inf AGR ?a VFORM ?v*) (VP **VFORM** inf)
7. (VP **AGR** ?a **VFORM** ?v) →
   (*V SUBCAT _np_vp:inf AGR ?a VFORM ?v*) NP (VP **VFORM** inf)
8. (VP **AGR** ?a **VFORM** ?v) →
   (*V SUBCAT _adjp AGR ?a VFORM ?v*) ADJP
9. (VP **SUBCAT** inf **AGR** ?a **VFORM** inf) →
   (*TO AGR ?a VFORM inf*) (VP **VFORM** base)
10. ADJP → *ADJ*
11. ADJP → (*ADJ SUBCAT _inf*) (VP **VFORM** inf)

Grammar 4.8 The expanded grammar showing all features

1. S[–inv] → (NP **AGR** ?a) (*VP*[{*pres past*}] *AGR ?a*)
2. NP → (ART **AGR** ?a) (*N AGR ?a*)
3. NP → *PRO*
4. VP → *V*[*_none*]
5. VP → *V*[*_np*] NP
6. VP → *V*[*_vp:inf*] VP[inf]
7. VP → *V*[*_np_vp:inf*] NP VP[inf]
8. VP → *V*[*_adjp*] ADJP
9. VP[inf] → *TO* VP[base]
10. ADJP → *ADJ*
11. ADJP → *ADJ*[*_vp:inf*] VP[inf]

Head features for S, VP: **VFORM**, **AGR**
Head features for NP: **AGR**

Grammar 4.7 A simple grammar in abbreviated form

2.8 Augmented Transition Networks

The ATN (augmented transition network) is produced by adding new features to a recursive transition network. Features in an ATN are traditionally called registers. Constituent structures are created by allowing each network to have a set of registers. Each time a new network is pushed, a new set of registers is created. As the network is traversed, these registers are set to values by actions associated with each arc. When the network is popped, the registers are assembled to form a constituent structure, with the CAT slot being the network name.

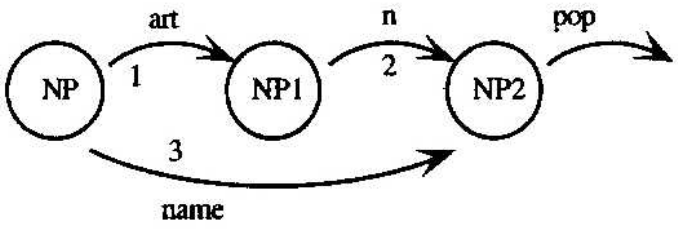
Consider Grammar 4.11 is a simple NP network. The actions are listed in the table below the network. ATNs use a special mechanism to extract the result of following an arc. When a lexical arc, such as arc 1, is followed, the constituent built from the word in the input is put into a special variable named "*". The action

DET := *

then assigns this constituent to the DET register. The second action on this arc,

AGR := AGR*

assigns the AGR register of the network to the value of the AGR register of the new word (the constituent in "*"). Agreement checks are specified in the tests. A test is an expression that succeeds if it returns a nonempty value and fails if it returns the empty set or nil.

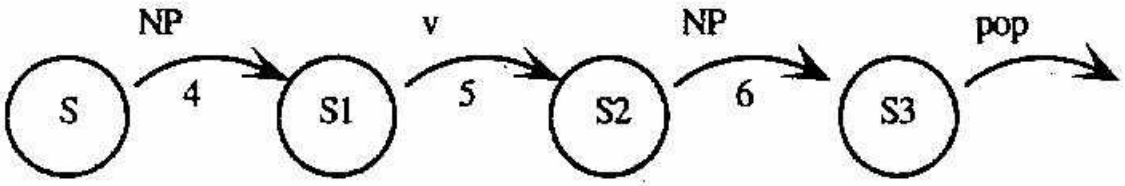


Arc	Test	Actions
1	none	DET := * AGR := AGR*
2	AGR ∩ AGR*	HEAD := * AGR := AGR ∩ AGR*
3	none	NAME := * AGR := AGR*

Grammar 4.11 A simple NP network

If a test fails, its arc is not traversed. The test on arc 2 indicates that the arc can be followed only if the AGR feature of the network has a non-null intersection with the AGR register of the new word (the noun constituent in "*").

Features on push arcs are treated similarly. The constituent built by traversing the NP network is returned as the value "*". Thus in Grammar 4.12, the action on the arc from S to S1,
SUBJ := *
would assign the constituent returned by the NP network to the register SUBJ. The test on arc 2 will succeed only if the AGR register of the constituent in the SUBJ register has a non-null intersection with the AGR register of the new constituent (the verb). This test enforces subject-verb agreement.



Arc	Test	Actions
4	none	SUBJ := *
5	$AGR_{SUBJ} \cap AGR_*$	MAIN-V := * $AGR := AGR_{SUBJ} \cap AGR_*$
6	none	OBJ := *

Grammar 4.12 A simple S network

ATNs Tracing Example:

The following Figure 4.13 Trace tests and actions used with "1 The 2 dog 3 saw 4 Jack 5"
With the lexicon in Section 4.3, the ATN accepts the following sentences:
The dog cried.
The dogs saw Jack.
Jack saw the dogs.
Consider an example. A trace of a parse of the sentence "The dog saw Jack" is shown in Figure 4.13. It indicates the current node in the network, the current

Trace of S Network

Step	Node	Position	Arc Followed	Registers Set
1.	S	1	arc 4 succeeds (for recursive call see trace below)	**SUBJ** ← (NP **DET** the **HEAD** dog **AGR** 3s)
5.	S1	3	arc 5 (checks if 3p ∩ 3p)	**MAIN-V** ← saw **AGR** ← 3p
6.	S2	4	arc 6 (for recursive call trace, see below)	**OBJ** ← (NP **NAME** Jack **AGR** 3s)
9.	S3	5	pop arc succeeds	returns (S **SUBJ** (NP **DET** the **HEAD** dog **AGR** 3s) **MAIN-V** saw **AGR** 3p **OBJ** (NP **NAME** Jack **AGR** 3s))

Trace of First NP Call: Arc 4

Step	Node	Position	Arc Followed	Registers Set
2.	NP	1	1	**DET** ← the **AGR** ← {3s 3p}
3.	NP1	2	2 (checks if {3s 3p} ∩ 3p)	**HEAD** ← dog
4.	NP2	3	pop	returns (NP **DET** the **HEAD** dog **AGR** 3s)

Trace of Second NP Call: Arc 6

Step	Node	Position	Arc Followed	Registers Set
7.	NP	4	3	**NAME** ← John **AGR** ← 3s
8.	NP2	5	pop	returns (NP **NAME** John **AGR** 3s)

Figure 4.13 Trace tests and actions used with $_1$ *The* $_2$ *dog* $_3$ *saw* $_4$ *Jack* $_5$

2.9 Bayes' Theorem:

Bayes theorem is also known as the Bayes Rule or Bayes Law. It is used to determine the conditional probability of event A when event B has already happened. The general statement of Bayes' theorem is "The conditional probability of an event A, given the occurrence of another event B, is equal to the product of the event of B, given A and the probability of A divided by the probability of event B." i.e.

$$P(A|B) = P(B|A)P(A) / P(B)$$

where,
***P(A)** and **P(B)** are the probabilities of events A and B*
***P(A|B)** is the probability of event A when event B happens*
***P(B|A)** is the probability of event B when A happens*

Bayes Theorem Statement

Bayes' Theorem for n set of events is defined as,

Let $E_1, E_2, \ldots, E_n$ be a set of events associated with the sample space S, in which all the events $E_1, E_2, \ldots, E_n$ have a non-zero probability of occurrence. All the events $E_1, E_2, \ldots, E$ form a partition of S. Let A be an event from space S for which we have to find probability, then according to Bayes' theorem,

$$P(E_i|A) = P(E_i)P(A|E_i) / \sum P(E_k)P(A|E_k)$$
$$\text{for } k = 1, 2, 3, \ldots, n$$

Terms Related to Bayes Theorem

As we have studied about Bayes theorem in detail, let us understand the meanings of a few terms related to the concept which have been used in the Bayes theorem formula and derivation:

Conditional Probability

The probability of an event A based on the occurrence of another event B is termed conditional Probability. It is denoted as **P(A|B)** and represents the probability of A when event B has already happened.

Joint Probability

When the probability of two more events occurring together and at the same time is measured it is marked as Joint Probability. For two events A and B, it is denoted by joint probability is denoted as, **P(A∩B).**

Random Variables

Real-valued variables whose possible values are determined by random experiments are called random variables. The probability of finding such variables is the experimental probability.

2.10 Shannon's Spelling Game

Competent spellers are good at recognizing **common spelling patterns**. This enables them to predict how any sound might be spelt because they know that there are only a limited number of options.

For example, if they hear an "o" sound, as in hope, they will consider -oa-; -oe-; or -o- followed, one consonant later, by the magic e

boat toe cope

Shannon's game helps to develop this kind of awareness. It's similar to Hangman except that the letters have to be guessed **in sequence**.

- Start by writing the first letter of a word.
- Then put down dashes to represent the other letters.
- Allow ten guesses for the next letter. If there is no correct guess, put the letter in and go on to the next.
- Continue until the whole word is completed.
- So, for example:
- q-------
  qu------
  que-----
  ques----
  quest---
  questi--
  questio-
  question

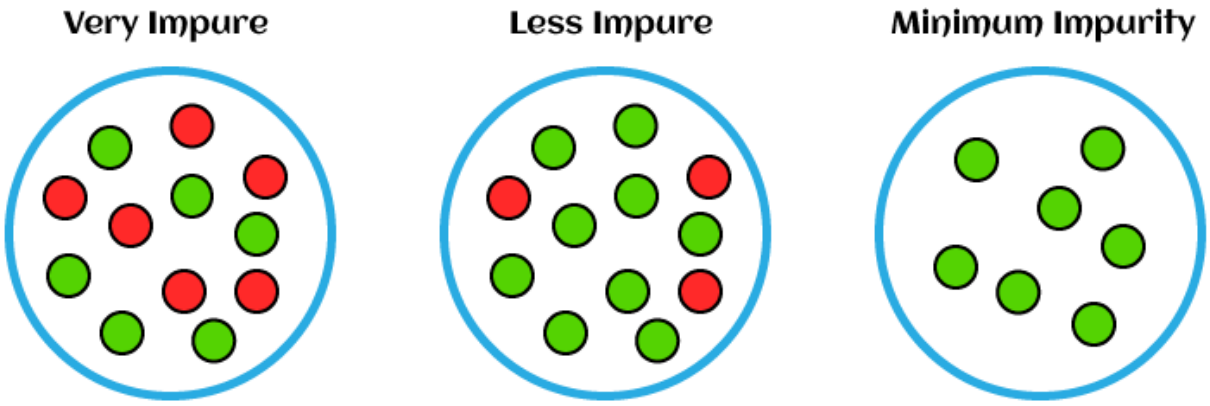
Sometimes it helps to have the alphabet written out in front of the players.

As players become more competent they are able to succeed with far fewer guesses. The game provides an ideal basis for parents to discuss the possible choices at any particular stage.

(The above is based on a page from my website Spelling it Right at www.spellitright.talktalk.net where you will find many other spelling topics)

2.11 Entropy and Cross Entropy

Entropy is defined as the randomness or measuring the disorder of the information being processed in Machine Learning. Further, in other words, we can say that **entropy is the machine learning metric that measures the unpredictability or impurity in the system**.



Entropy is a slippery concept in physics, but is quite straightforward in information theory. Suppose you have a process (like a language *L* that generates words). At each step in the process, there is some probability *p* that the thing that happened (the event) was going to happen. The amount of **surprisal** is –log(*p*) where the logarithm is taken in any base you want (equivalent to changing units). Low probability events have high surprisal. Events that were certain to happen (*p*=1) have 0 surprisals. Events that are impossible (*p*=0) have infinity surprisal.

The entropy is the expected value of the surprisal across all possible events indexed by *i*:

$$H(p) = -\sum_i p_i \log p_i$$

Entropy of a probability distribution p

So, the entropy is the average amount of surprise when something happens.

Entropy always lies between 0 and 1, however depending on the number of classes in the dataset, it can be greater than 1.

Entropy in base 2 is also optimal number of bits it takes to store the information about what happened, by Claude Shannon's source coding theorem. For example if I told you that a full-length tweet of 280 characters had an entropy of 1 bit per character, that means that, by the laws of mathematics, no matter what Twitter does, they will always have to have 280 bits (35 bytes) of storage for that tweet in their database.

In the context of our language model, we'll have to make one tweak. Given that we are interested in sentences *s* (sequences of events) of length *n*, we'll define the entropy rate per word (event) as:

$$H_n(L) = -\frac{1}{n}\sum_{s\in L} L(s)\log L(s)$$

where the sum is over all sentences of length *n* and L(s) is the probability of the sentence. Finally, a technical point: we want to define the entropy of the language L (or language model M) regardless of sentence length *n*. So finally we define

$$H(L) = \lim_{n\to\infty} -\frac{1}{n}\sum_{s\in L} L(s)\log L(s)$$

Final definition of entropy for a language (model)

The Shannon-McMillan-Breiman Theorem

Under anodyne assumptions[3] the entropy simplifies even further. The essential insight is that, if we take a long enough string of text, each sentence occurs in proportion to its probability anyways. So there is no need to sum over possible sentences. We get:

$$H(L) = \lim_{n\to\infty} -\frac{1}{n}\log L(s)$$

Simplification of entropy with the Shannon-McMillan-Breiman Theorem

This tells us that we can just take a large (n is big) text instead of trying to sample from diverse texts.

Cross-Entropy:

Suppose we mistakenly think that our language model M is correct. Then we observe text generated by the actual language L without realizing it. The *cross-entropy* H(L,M) is what we measure the entropy to be

$$H(L,M) = \lim_{n\to\infty} -\frac{1}{n}\sum_{s\in L} L(s)\log M(s)$$

$$= \lim_{n\to\infty} -\frac{1}{n}\log M(s)$$

Cross entropy for our language model M

Where the second line again applies the Shannon-McMillan-Breiman theorem.
Crucially, this tells us we can estimate the cross-entropy H(L,M) by just measuring log M(s) for a random sample of sentences (the first line) or a sufficiently large chunk of text (the second line).

The Cross-Entropy is Bounded by the True Entropy of the Language

The cross-entropy has a nice property that **H(L) ≤ H(L,M)**. Omitting the limit and the normalization 1/n in the proof:

$$\begin{aligned} H(L) &= -\sum L(s)\log\left[L(s)\right] \\ &= -\sum L(s)\log\left[\frac{L(s)}{M(s)}M(s)\right] \\ &= -\left(\sum L(s)\log M(s)\right) - \left(\sum L(s)\log\frac{L(s)}{M(s)}\right) \\ &= H(L,M) - D_{\mathrm{KL}}(L||M) \end{aligned}$$

In the third line, the first term is just the cross-entropy (remember the limits and 1/n terms are implicit). The second term is the Kullback-Leibler divergence (or KL-divergence). By Gibbs' inequality the KL-divergence is non-negative and is 0 only if the models L and M are the same. The KL-divergence is sort of like a distance measure.

PBR VITS (AUTONOMOUS)

IVB.TECH CSE-IOT

NATURAL LANGUAGE PROCESSING

TEXT BOOK: James Allen, **Natural Language Understanding**, 2nd Edition, 2003, Pearson Education

Course Instructor: Dr KV Subbaiah, M.Tech, Ph.D, Professor, Dept. of CSE

UNIT–III Grammars for Natural Language

Grammars for Natural Language, Movement Phenomenon in Language, Handling questions in Context Free Grammars, Hold Mechanisms in ATNs, Gap Threading, Human Preferences in Parsing, Shift Reduce Parsers, Deterministic Parsers.

3.1 Movement Phenomena in Language

Many sentence structures appear to be simple variants of other sentence structures. In some cases, simple words or phrases appear to be locally reordered; sentences are identical except that a phrase apparently is moved from its expected position in a basic sentence. This section explores techniques for exploiting these generalities to cover questions in English.

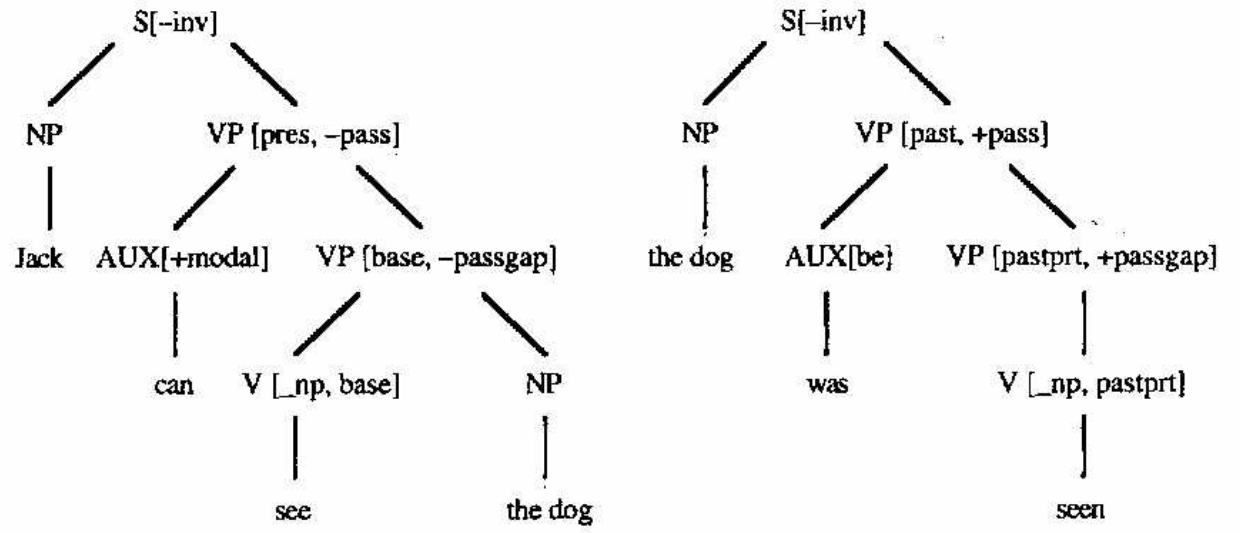


Figure 5.4 An active and a passive form sentence

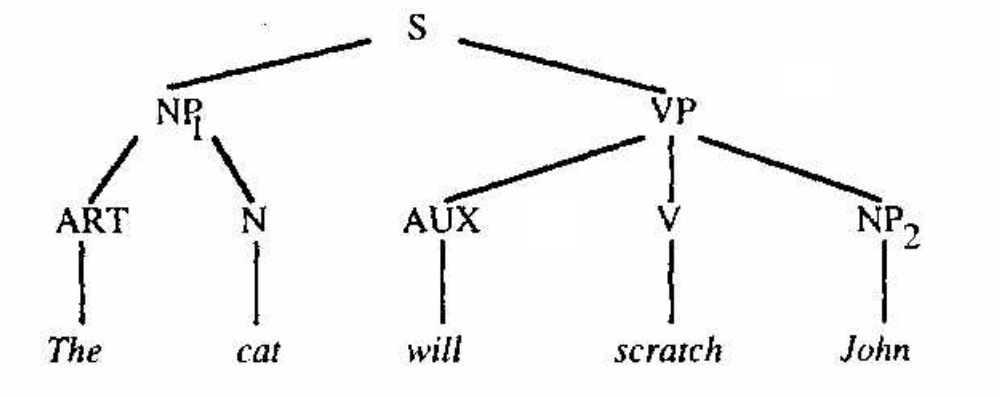
As a starting example, consider the structure of yes/no questions and how they relate to their assertional counterpart. In particular, consider the following examples:

Jack is giving Sue a back rub.
He will run in the marathon next year.
Is Jack giving Sue a back rub?
Will he run in the marathon next year?
As you can readily see, yes/no questions appear identical in structure to their assertional counterparts except that the subject NPs and first auxiliaries have swapped positions. If there is no auxiliary in the assertional sentence, an auxiliary of root "*do"*, in the appropriate tense, is used:
John went to the store. Henry goes to school every day.
Did John go to the store? Does Henry go to school every day?
Taking a term from linguistics, this rearranging of the subject and the auxiliary is called subject-aux inversion.

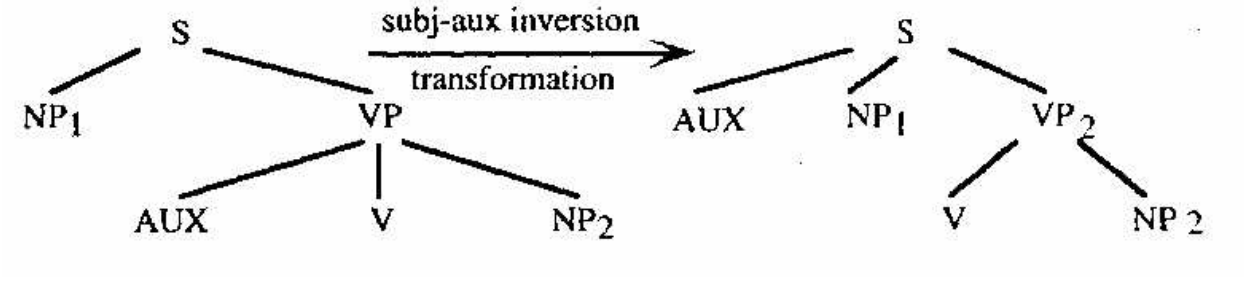
On the other hand, if you are interested in how it is done, you might ask one of the following questions:
How will the fat man put the book in the corner?
In what way will the fat man put the book in the corner?
If you are interested in other aspects, you might ask one of these questions:
What will the fat man angrily put in the corner?
Where will the fat man angrily put the book?
In what corner will the fat man angrily put the book?
What will the fat man angrily put the book in?
Each question has the same form as the original assertion, except that the part being questioned is removed and replaced by a wh-phrase at the beginning of the sentence. In addition, except when the part being queried is the subject NP, the subject and the auxiliary are apparently inverted, as in yes/no questions. This similarity with yes/no questions even holds for sentences without auxiliaries. In both cases, a "*do"* auxiliary is inserted:
I found a bookcase.
Did I find a bookcase?
What did I find?

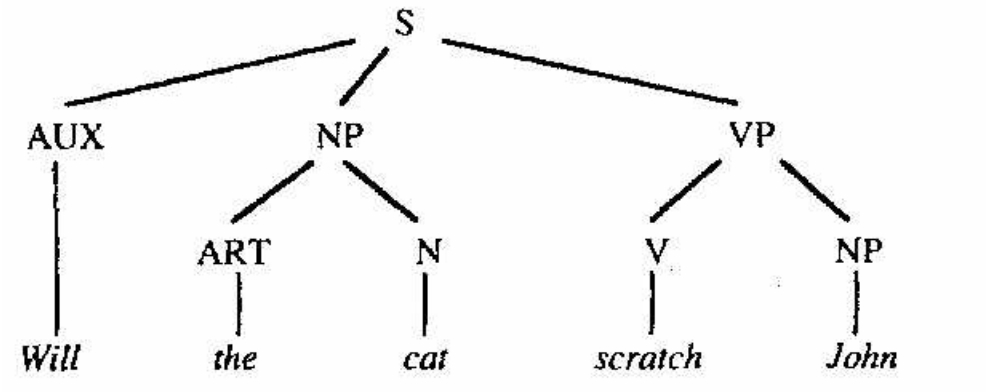
The term *movement* arose in transformational grammar (TG). TG posited two distinct levels of structural representation: surface structure, which corresponds to the actual sentence structure, and deep structure. A CFG generates the deep structure, and a set of transformations map the deep structure to the surface structure.

For example, the deep structure of "*Will the cat scratch John?"* would be:



The yes/no question is then generated from this deep structure by a transformation expressed schematically as follows





3.2 Handling Questions in Context-Free Grammars

The main goal is to extend a context-free grammar minimally so that it can **handle questions**. We want to reuse as much of the original grammar as possible. For yes/no questions, this is easily done. We can extend following Grammar with one rule that allows an auxiliary before the first NP and handles most examples:

1. S[–inv] → (NP **AGR** ?a) (*VP* [{*pres past*}] ***AGR*** *?a*)
2. NP → (ART **AGR** ?a) (*N* ***AGR*** *?a*)
3. NP → *PRO*
4. VP → *V*[*_none*]
5. VP → *V*[*_np*] NP
6. VP → *V*[*_vp:inf*] VP[inf]
7. VP → *V*[*_np_vp:inf*] NP VP[inf]
8. VP → *V*[*_adjp*] ADJP
9. VP[inf] → *TO* VP[base]
10. ADJP → *ADJ*
11. ADJP → *ADJ*[*_vp:inf*] VP[inf]

Head features for S, VP: **VFORM, AGR**
Head features for NP: **AGR**

Grammar 4.7 A simple grammar in abbreviated form

S [+inv] -> (AUXAGR ?a SUBCAT ?v) (NP AGR ?a) (VP VFORM ?v)
This enforces subject-verb agreement between the AUX and the subject NP, and ensures that the VP has the right VFORM to follow the AUX. This one rule is all that is needed to handle yes/no questions, and all of the original grammar tor assertions can be used directly for yes/no questions.

An algorithm for automatically adding GAP features to a grammar is shown in Figure 5.5. Note that it does not modify any rule that explicitly sets the GAP feature already, allowing the grammar designer to introduce rules that do not follow the conventions encoded in the algorithm. In particular, the rule for subject-aux inversion cannot allow the gap to propagate to the subject NP.

For each rule $Y \rightarrow X_1 \ldots H_i \ldots X_n$ with head constituent H_i

1. If the rule specifies a GAP feature in some constituent already, then skip.
2. If the head H_i is not a lexical category, then add a GAP feature to the head and the mother, and –GAP to the other subconstituents, producing a rule of form:
   $(Y\ \mathbf{GAP}\ ?g) \rightarrow (X_1\ \mathbf{GAP}-) \ldots (H_i\ \mathbf{GAP}\ ?g) \ldots (X_n\ \mathbf{GAP}-)$
3. If the head H_i is a lexical category, then for each nonlexical subconstituent X_j, add a rule of the form:
   $(Y\ \mathbf{GAP}\ ?g) \rightarrow (X_1\ \mathbf{GAP}-) \ldots (X_j\ \mathbf{GAP}\ ?g) \ldots (X_n\ \mathbf{GAP}-)$

Figure 5.5 An algorithm for adding GAP features to a grammar

Using this procedure, a new grammar can be created that handles gaps. All that is left to do is analyze where the fillers for the gaps come from. In wh questions, the fillers are typically NPs or PPs at the start of the sentence and are identified by a new feature WH that identifies a class of phrases that can introduce questions. The WH feature is signaled by words such as *who, what, when, where, why,* and *how* (as in *how many* and *how carefully).* These words fall into several different grammatical categories, as can be seen by

considering what type of phrases they replace. In particular, *who, whom,* and *what* can appear as pronouns and can be used to specify simple NPs:

Who ate the pizza?

What did you put the book in?

The words "*what"* and "*which"* may appear as determiners in noun phrases, as in

What book did he steal?

Words such as "*where"* and "*when"* appear as prepositional phrases:

Where did you put the book?

The word "*how"* acts as an adverbial modifier to adjective and adverbial phrases:

How quickly did he run?

Finally, the word "*whose"* acts as a possessive pronoun:

Whose book did you find?

A simple NP and PP grammar handling wh-words

1. (NP **POSS** ?p **WH** ?w) → (*PRO **POSS** ?p **WH** ?w*)
2. (NP **WH** ?w) → (DET **WH** ?w **AGR** ?a) (*CNP **AGR** ?a*)
3. CNP → *N*
4. CNP → ADJ *N*
5. DET → *ART*
6. (DET **WH** ?w) → (*NP[+POSS] **WH** ?a*)
7. (DET **WH** ?w) → (*QDET **WH** ?w*)
8. (PP **WH** ?w) → *P* (NP **WH** ?w)
9. (PP **WH** ?w) → (*PP-WRD **WH** ?w*)

Head feature for NP, DET and CNP: AGR
Head feature for PP: PFORM

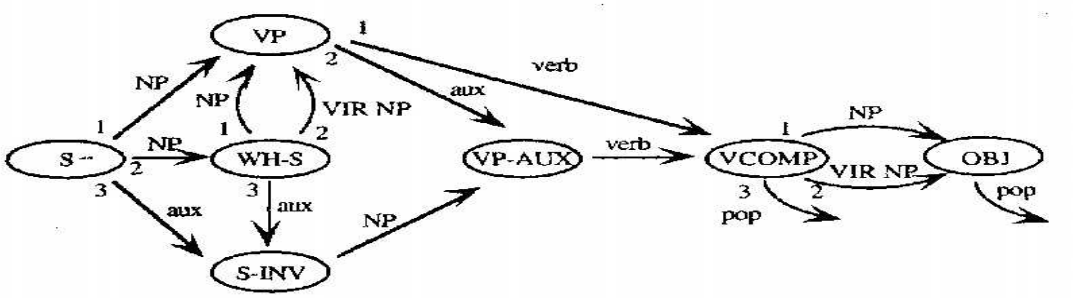
Grammar 5.7 A simple NP and PP grammar handling wh-words

3.3 Hold Mechanisms in ATNs

Another technique for handling movement was first developed with the ATN framework. A data structure called the hold list maintains the constituents that are to be moved. Unlike GAP features, more than one constituent may be on the hold list at a single time. Constituents are added to the hold list by a new action on arcs, the hold action, which takes a constituent and places it on the hold list.

The hold action can store a constituent currently in a register (for example, the action HOLD SUBJ holds the constituent that is in the SUBJ register). To ensure that a held constituent is always used to fill a gap, an ATN system does not allow a pop arc to succeed from a network until any constituent held by an action on an arc in that network has been used. That is, the held constituent must have been used to fill a gap in the current constituent or in one of its subconstituents.

Finally, you need a mechanism to detect and fill gaps. A new arc called VIR (for virtual) that takes a constituent name as an argument can be followed if a constituent of the named category is present on the hold list. If the arc is followed successfully, the constituent is removed from the hold list and returned as the value of the arc in the identical form that a PUSH arc returns a constituent.



Arc	Test	Actions
S/1		INV := – SUBJ := *
S/2	WH* ∩ [Q R]	WH := WH* HOLD *
S/3	VFORM* ∩ {past pres}	INV := + AUX := *
WH-S/1		SUBJ := *
WH-S/2		SUBJ := *
WH-S/3	VFORM* ∩ {past pres}	AUX := *
VP/1	AGR$_{SUBJ}$ ∩ AGR* VFORM* ∩ {past pres}	MAIN-V := *
VP/2	AGR$_{SUBJ}$ ∩ AGR* VFORM* ∩ {past pres}	AUX := *
S-INV/1	AGR$_{AUX}$ ∩ AGR*	SUBJ := *
VP-AUX/1	SUBCAT$_{AUX}$ ∩ FORM*	MAIN-V := *
VCOMP/1	SUBCAT$_{MAIN-V}$ ∩ _np	OBJ := *
VCOMP/2	SUBCAT$_{MAIN-V}$ ∩ _np	OBJ := *
VCOMP/3	SUBCAT$_{MAIN-V}$ ∩ _none	

Grammar 5.14 An S network for questions and relative clauses

The network is organized so that all +INV sentences go through node S-INV, while all -INV sentences go through node VP. All wh-questions and relative clauses go through node WH-S and then are redirected back into the standard network based on whether or not the sentence is inverted. The initial NP is put on the hold list when arc Sf2 is followed. For noninverted questions, such as *Who ate the pizza?,* and for relative clauses in the NP of form *the man who ate the pizza,* the held NP is immediately used by the VIR arc WH-S/2. For other relative clauses, as in the NP *the man who we saw,* arc WH-S/1 is used to accept the subject in the relative clause, and the held NP is used later on arc VCOMP/2. For inverted questions, such as *Who do I see?,* arc WH-S/3 is followed to accept the auxiliary, and the held NP must be used later.

This network only accepts verbs with SUBCAT values _none and _np but could easily be extended to handle other verb complements. When extended, there would be a much wider range of locations where the held NP might be used. Figure 5.15 shows a trace of the sentence "*The man who we saw cried".* In parsing the relative clause, the relative pronoun *who* is held in step *5* and used by a VIR arc in step 8.

Note that this ATN would not accept **Who did the man see the boy?,* as the held constituent *who* is not used by any VIR arc; thus the pop arc from the S network cannot be taken. Similarly, **The man who the boy cried ate the pie* is unacceptable, as the relative pronoun is not used by a VIR arc in the S network that analyzes the relative clause.

3.4 Gap Threading

A third method for handling gaps combines aspects of both the GAP feature approach and the hold list approach. This technique is usually called gap threading. It is often used in logic grammars, where two extra argument positions are added to each predicate—one argument for a list of fillers that might be used in the current constituent, and one for the resulting list of fillers that were not used after the constituent is parsed. Thus the predicate

s (position-in, position-out, fillers-in, fillers-out)

is true only if there is a legal S constituent between *position-in* and *position-out* of the input. If a gap was used to build the 5, its filler will be present in *fillers-in,* but not in *fillers-out.* For example, an S constituent with an NP gap would correspond to the predicate s(ln, Out, (NP], nil). In cases where there are no gaps in a constituent, the *fillers-in* and *fillers-out* will be identical.

Consider an example dealing with relative clauses. The rules required are shown in Grammar 5.16. The various feature restrictions that would be needed to enforce agreement and subcategorization are not shown so as to keep the example simple.

A logical Grammar for GAP threading:

```
1.  s(In, Out, FillersIn, FillersOut) :- np(In, In1, FillersIn, Fillers1),
                                        vp(In1, Out, Fillers1, FillersOut)
2.  vp(In, Out, FillersIn, FillersOut) :- v(In, In1)
3.  vp(In, Out, FillersIn, FillersOut) :- v(In, In1), np(In1, Out, FillersIn, FillersOut)
4.  np(In, Out, Fillers, Fillers) :- art(In, In1), cnp(In1, Out)
5.  np(In, Out, Fillers, Fillers) :- pro(In, Out)
6.  cnp(In, Out) :- n(In, In1), np-comp(In1, Out)
7.  np-comp(In, In) :-
    (This covers the case where there is no NP complement.)
8.  np-comp(In, Out) :- rel-intro(In, In1, Filler),
                        s(In1, Out, (Filler nil), nil)
    (Here we hold the Rel-Intro constituent, and must use it in the following S.)
9.  rel-intro(In, Out, [NP]) :- relpro(In, Out)
    (where relpro accepts any pronoun with WH feature R)
10. np(In, In, [NP | Fillers], Fillers) :-
    (This rule builds an empty np from a filler.)
```

Grammar 5.16 A logic grammar using gap threading

To see these rules in use, consider the parse of the sentence *The man who we saw cried* in Figure 5.17. The relative clause is analyzed starting with step 7. Using rule 9, the word *who* is recognized as a relative pronoun, and the variable Filler is bound to the list [NP]. This filler is then passed into the embedded S (step 9), to the NP (step 10), and then on to the VP (step 12), since it is not used in the NP. From there it is passed to the NP predicate in step 14, which uses the filler according to rule 10. Note that no other NP rule could have applied at this point, because the filler must be used since the FillersOut variable is nil. Only rules that consume the filler can apply. Once this gap is used, the entire NP from positions 1 to 6 has been found and the rest of the parse is straightforward.

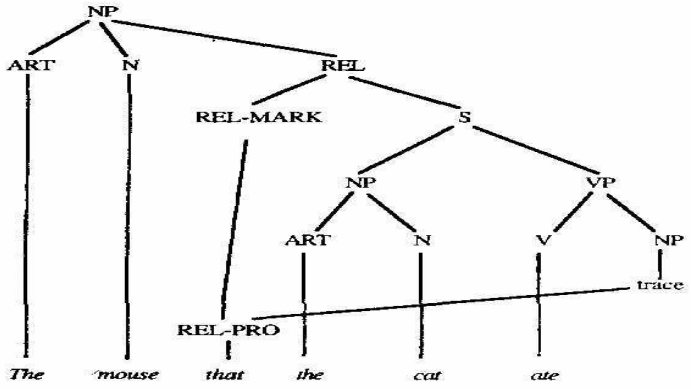


Figure 5.18 A parse tree in extraposition grammar

3.5 Human Preferences in Parsing

Generally the people will know simple rules to parse sentences. The Psycholinguists have conducted many investigations into parsing using a variety of techniques. These studies have revealed some general principles concerning how people resolve ambiguity.

1.1	S → NP VP	1.4	NP → ART N
1.2	VP → V NP PP	1.5	NP → NP PP
1.3	VP → V NP	1.6	PP → P NP

Grammar 6.1 A simple CFG

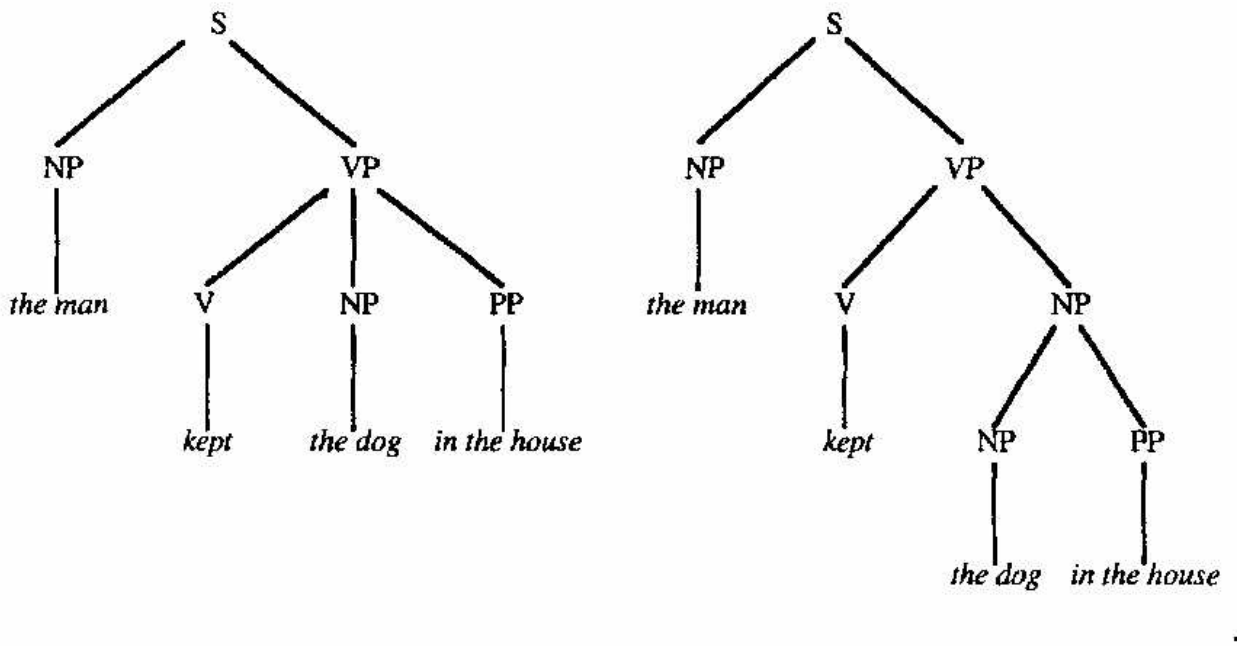


Figure 6.2 The interpretation on the left is preferred by the minimal attachment principle

The most basic result from these studies is that people do not give equal weight to all possible syntactic interpretations. This can be illustrated by sentences that are temporarily ambiguous, which cause a conscious feeling of having pursued the wrong analysis, as in the sentence "*The raft floated down the river sank".* When you read the word "*sank"* you realize that the interpretation you have constructed so far for the sentence is not correct. In the literature, such sentences are often called "**garden-path"** sentences, based on the expression about leading someone down a garden path. Here are a few of the general principles that appear to predict when garden paths will arise.

Minimal Attachment

The most general principle is called the **minimal attachment** principle**,** which states that there is a preference for the syntactic analysis that creates the least number of nodes in the parse tree. Thus, given Grammar 6.1, the sentence "*The man kept the dog in the house"* would be interpreted with the PP "*in the house"* modifying the verb rather than the NP "*the dog".* These two interpretations are shown in Figure 6.2. The interpretation with the PP attached to the VP is derived using rules 1.1, 1.2, and 1.6 and three applications of rule 1.4 for the NPs. The parse tree has a total of 14 nodes. The interpretation with the PP attached to the NP is derived using rules 1.1,

1.3, 1.5, and 1.6 and three applications of rule 1.4, producing a total of 15 nodes in the parse tree. Thus this principle predicts that the first interpretation is preferred, which probably agrees with your intuition.

This principle appears to be so strong that it can cause certain sentences to be almost impossible to parse correctly. One example is the sentence

We painted all the walls with cracks.

which, against all common sense, is often read as meaning that cracks were painted onto the walls, or that cracks were somehow used as an instrument to paint the walls. Both these anomalous readings arise from the PP being attached to the VP *(paint)* rather than the NP *(the walls).* Another classic example is the sentence

The horse raced past the barn fell.

which has a reasonable interpretation corresponding to the meaning of the sentence "*The horse that was raced past the barn fell".* In the initial sentence, however, creating a reduced relative clause when the word "*raced"* is encountered introduces many more nodes than the simple analysis where "*raced"* is the main verb of the sentence. Of course, this second interpretation renders the sentence unanalyzable when the wordfell is encountered.

Right Association

The second principle is called right association or late closure. This principle states that, all other things being equal, new constituents tend to be interpreted as being part of the current constituent under construction (rather than part of some constituent higher in the parse tree). Thus, given the sentence

George said that Henry left in his car.

the preferred interpretation is that Henry left in the car rather than that George spoke in the car. Both interpretations are, of course, syntactically acceptable analyses. The two interpretations are shown in Figure 6.3. The former attaches the PP to the VP immediately preceding it, whereas the latter attaches the PP to the VP higher in the tree. Thus the right association principle prefers the former. Similarly, the preferred interpretation for the sentence "*I thought it would rain yesterday"* is that yesterday was when it was thought to rain, rather than the time of the thinking.

Lexical Preferences

In certain cases the two preceding principles seem to conflict with each other. In the sentence "*The man kept the dog in the house"*, the principle of right association appears to favor the interpretation in which the PP modifies the dog, while the minimal attachment principle appears to favor the PP modifying the VP. You might suggest that minimal attachment takes priority over right association in such cases; however, the relationship appears to be more complex than that. Consider the sentences

1. I wanted the dog in the house.
2. I kept the dog in the house.
3. I put the dog in the house.

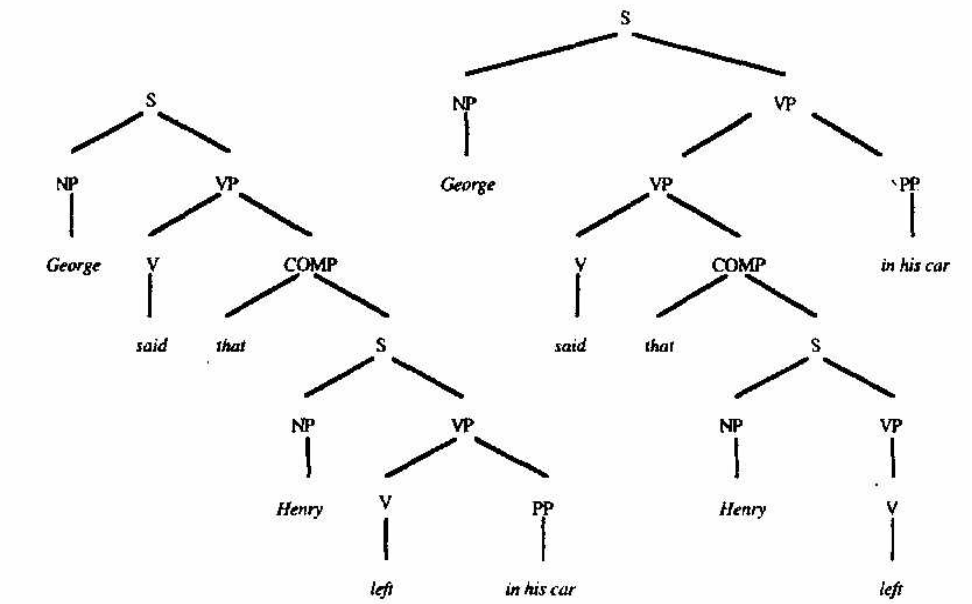


Figure 6.3 Two interpretations of *George said that Henry left in his car.*

The PP "*in the house"* in sentence I seems most likely to be modifying "*dog"* (al though the other interpretation is possible, as in the sense "*I wanted the dog to be in the house").* In sentence 2, the PP seems most likely to be modifying the VP (although modifying the NP is possible, as in "*I kept the dog that was in the house").* Finally, in sentence 3, the PP is definitely attached to the VP, and no alternative reading is possible.

These examples demonstrate that lexical items, in this case the verb used, can influence parsing preferences. In many cases, the lexical preferences will override the preferences based on the general principles. For example, if a verb subcategorizes for a prepositional phrase, then some PP must be attached to the VP. Other PP5 might also be identified as having a strong preference for attachment within the VP. If neither of these cases holds, the PP will be attached according to the general principles.
Thus, for the preceding verbs, "*want"* has no preference for any PPs, whereas "*keep"* might prefer PPs with prepositions "*in", "on",* or "*by"* to be attached to the VP. Finally, the verb "*put"* requires (subcategorizes for) a PP beginning with "*in", "on", "by",* and so on, which must be attached to the VP.

2.1 $S \rightarrow NP\ VP$	2.3 $VP \rightarrow AUX\ V\ NP$
2.2 $NP \rightarrow ART\ N$	2.4 $VP \rightarrow V\ NP$

Grammar 6.4 A simple grammar with an AUX/V ambiguity

3.6 Shift Reduce Parsers

The Shift Reduce parsers are used to improve the efficiency of parsers in NLP. The uncertainty is passed forward through the parse to the point where the input eliminates all but one of the possibilities. The efficiency of the technique described in this section arises from the fact that all the possibilities are considered in advance, and the information is stored in a table that controls the parser, resulting in parsing algorithms that can be much faster than described thus far.

These techniques were developed for use with unambiguous context-free grammars - grammars for which there is at most one interpretation for any given sentence. While this constraint is reasonable for programming languages, it is clear that there is no unambiguous grammar for natural language. But these techniques can be extended in various ways to make them applicable to natural language parsing.

Specifying the Parser State

Consider using this approach on the small grammar in Grammar 6.4. The technique involves predetermining all possible parser states and determining the transitions from one state to another. A parser state is defined as the complete set of dotted rules (that is, the labels on the active arcs in a chart parser) applicable at that position in the parse. It is complete in the sense that if a state contains a rule of the form **Y -> ... o X ...**, where **X** is a nonterminal, then all rules for **X** are also contained in the state. For instance, the initial state of the parser would include the rule

S -> o NP VP

as well as all the rules for NP, which in Grammar 6.4 is only

NP -> o ART N

Thus the initial state, S0, could be summarized as follows:

Initial State S0: S -> o NP VP
NP -> o ART N

In other words, the parser starts in a state where it is looking for an NP to start building an S and looking for an ART to build the NP. What states could follow this initial state? To calculate this, consider advancing the dot over a terminal or a nonterminal and deriving a new state. If you pick the symbol
ART, the resulting state is

State S1: NP -> ART o N

If you pick the symbol NP, the rule is

S -> NP o VP

in the new state. Now if you expand out the VP to find all its possible starting symbols, you get the following:

State S2: S -> NP o VP
VP -> o AUX V NP
VP -> o V NP

Now, expanding SI, if you have the input N, you get a state consisting of a completed rule:

State S1': NP -> ART N o

Expanding 52, a V would result in the state

State S3: VP -> V o NP
NP -> o ART N

An AUX from S2 would result in the state

State S4: VP -> AUX o V NP

and a VP from 52 would result in the state

State S2': S -> NP VP o

Continuing from state 53 with an ART, you find yourself in state Si again, as you would also if you expand from S0 with an ART. Continuing from S3
with an NP, on the other hand, yields the new state

State S3': VP -> V NP o

Continuing from S4 with a V yields
State S5: VP -> AUX V o NP
NP -> o ART N

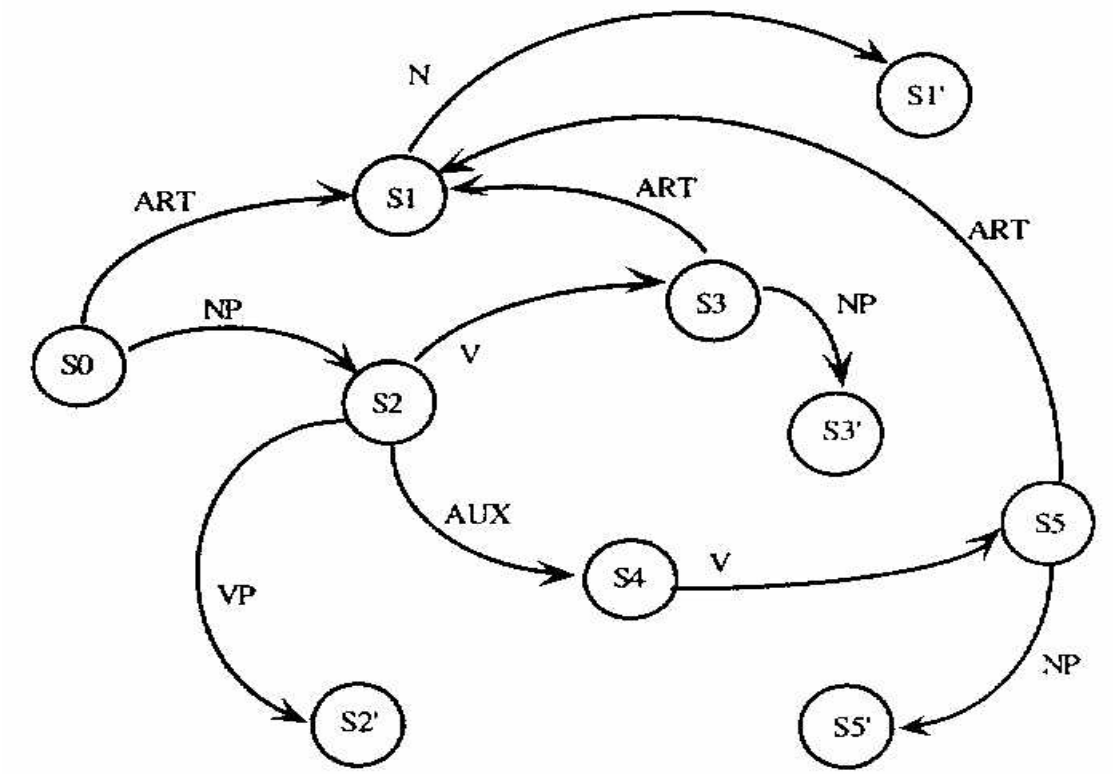


Figure 6.5 A transition graph derived from Grammar 6.1

and continuing from S5 with an ART would produce state S1 again. Finally, continuing from S5 with an NP would produce the state
State S5': VP -> AUX V NP o
Now that this process is completed, you can derive a transition graph that can be used to control the parsing of sentences, as is shown in Figure 6.5.

A Shift-Reduce Parser

These states can be used to control a parser that maintains two stacks: the **parse stack**, which contains parse states (that is, the nodes in Figure 6.5) and grammar symbols; and the input stack, which contains the input and some grammar symbols. At any time the parser operates using the information specified for the top state on the

parse stack. The states are interpreted as follows. The states that consist of a single rule with the dot at the far right-hand side, such as S2',

S -> NP VP o

indicate that the parser should rewrite the top symbols on the parse stack according to this rule. This is called a reduce action. The newly derived symbol (S in this case) is pushed onto the top of the input stack.

Any other state not containing any completed rules is interpreted by the transition diagram. If the top input symbol matches an arc, then it and the new state (at the end of the arc) are pushed onto the parse stack. This is called the **shift action.** Using this interpretation of states you can construct a table, called the oracle, that tells the parser what to do in every situation. The oracle for Grammar 6.4 is shown in Figure 6.6. For each state and possible input, it specifies the action and the next state. Reduce actions can be applied regardless of the next input, and the accept action only is possible when the input stack is empty (that is, the next symbol is the empty symbol e). The parsing algorithm for using an oracle is specified in Figure 6.7.

Consider parsing "*The man ate the carrot".* The initial state of the parser is

Parse Stack	**Input Stack**
(S0)	**(The man ate the carrot)**

Looking up the entry in the table in Figure 6.6 for state SO for the input ART (the category of the word *the),* you see a shift action and a move to state S1:

Parse Stack	**Input Stack**
(S1 ART S0)	**(man ate the carrot)**

Looking up the entry for state S1 for the input N, you see a shift action and a move to state S1:

Parse Stack	**Input Stack**
(S1' N S1 ART S0)	**(ate the carrot)**

Looking up the entry for state Si′, you then reduce by rule 2.2, which removes the Si,, N, 51, and ART from the parse stack and adds NP to the input stack:

Parse Stack	**Input Stack**
(S0)	**(NP ate the carrot)**

Again, consulting the table for state S0 with input NP, you now do a shift and move to state S2:

Parse Stack	**Input Stack**
(S2 NP S0)	**(ate the carrot)**

Next, the three remaining words all cause shifts and a move to a new state, ending up with the parse state:

Parse Stack	**Input Stack**
(S1' N S1 ART S3 V S2 NP S0)	**( )**

The reduce action by rule 2.2 specified in state S1' pops the N and ART from the stack (thereby popping S1 and S1' as well), producing the state:

Parse Stack	**Input Stack**
(S3 V S2 NP S0)	**(NP)**

You are now back at state S3, with an NP in the input, and after a shift to state S3', you reduce by rule 2.4, producing:

Parse Stack	**Input Stack**

This algorithm uses the following information:

- *Action*(S, W)—a function that maps a state and an input constituent to one of the values *shift, reduce i,* or *accept*
- *GoTo*(S, W)—a function that maps a state and an input constituent to a new state
- a parse stack of form $(S_n\ C_n\ \ldots\ S_1\ C_1\ S_0)$, where S_i are parse states and C_i are constituents
- an input stack of form $(W_1\ \ldots\ W_n)$, where W_i is a constituent symbol or word

The parser operates by continually executing the following steps until success or failure:

1. If *Action*$(S_n, W_1) =$ *Shift*, and *GoTo*$(S_n, W_1) = S$, then remove W_1 from the input stack and push it on the parse stack, and then push S onto the parse stack, resulting in the following stacks:

   parse stack: $(S\ W_1\ S_n\ C_n\ \ldots\ S_1\ C_1\ S_0)$
   input stack: $(W_2\ \ldots\ W_n)$

2. If *Action*$(S_n, W_1) =$ *Reduce i* and grammar rule i has n constituents on its right-hand side, then remove 2n elements from the parse stack, and push the left-hand side of rule i onto the input stack. For example, if rule i were NP → ART N, then the new state would be

   parse stack: $(S_{n-2}\ C_{n-2}\ \ldots\ S_1\ C_1\ S_0)$
   input stack: $(NP\ W_1\ \ldots\ W_n)$

3. If *Action*$(S_n, W_1) =$ *Accept*, then the parser has succeeded.
4. If *Action*(S_n, W_1) is not defined, then the parser has failed.

Figure 6.7 The parsing algorithm for a shift-reduce parser

State	Top Input Symbol	Action	GoTo
S0	ART	Shift	S1
S0	NP	Shift	S2
S0	S	Shift	S0'
S0'	ε	Succeed	-----
S1	N	Shift	S1'
S1'	----------	Reduce by rule 2.2	-----
S2	V	Shift	S3
S2	AUX	Shift	S4
S2	VP	Shift	S2'
S2'	----------	Reduce by rule 2.1	-----
S3	ART	Shift	S1
S3	NP	Shift	S3'
S3'	----------	Reduce by rule 2.4	-----
S4	V	Shift	S5
S5	ART	Shift	S1
S5	NP	Shift	S5'
S5'	----------	Reduce by rule 2.3	-----

Figure 6.6 The oracle for Grammar 6.4

3.7 Deterministic Parsers.

A deterministic parser can be built that depends entirely on matching parse states to direct its operation. Instead of allowing only shift and reduce actions, however, a richer set of actions is allowed that operates on an input stack called the buffer. (**The cat ate the fish**).

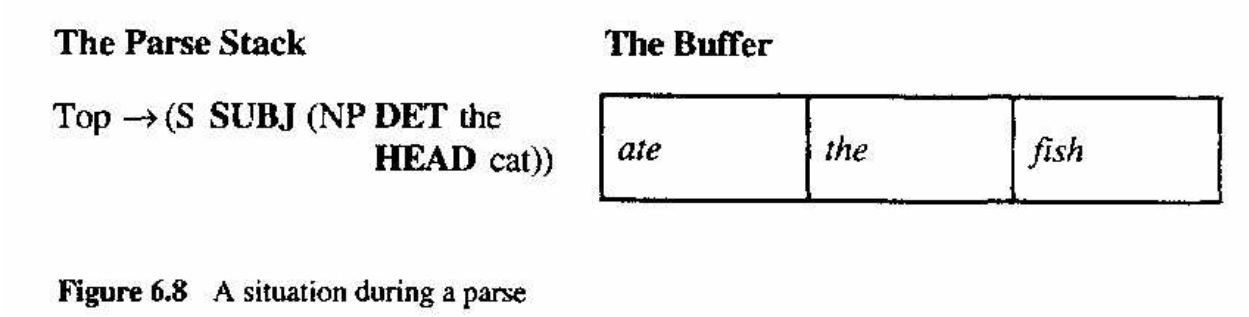


Figure 6.8 A situation during a parse

Rather than shifting constituents onto the parse stack to be later consumed by a reduce action, the parser builds constituents incrementally by attaching buffer elements into their parent constituent, an operation similar to feature assignment. Rather than shifting an NP onto the stack to be used later in a reduction S -> NP VP, an S constituent is created on the parse stack and the NP is attached to it. Specifically, this parser has the following operations:

- Create a new node on the parse stack (to push the symbol onto the stack)
- Attach an input constituent to the top node on the parse stack
- Drop the top node in the parse stack into the buffer

The drop action allows a completed constituent to be reexamined by the parser, which will then assign it a role in a higher constituent still on the parse stack. This technique makes the limited lookahead technique surprisingly powerful.

To get a feeling for these operations, consider the situation in Figure 6.8, which might occur in parsing the sentence "*The cat ate the fish*". Assume that the first NP has been parsed and assigned to the SUBJ feature of the S constituent on the parse stack. The operations introduced earlier can be used to complete the analysis. Note that the actual mechanism for deciding what operations to do has not yet been discussed, but the effect of the operations is shown here to provide intuition about the data structure. The operation

Attach to MAIN-V

would remove the lexical entry for *ate* from the buffer and assign it to the MAIN-V feature in the S on the parse stack. Next the operation

Create NP

would push an empty NP constituent onto the parse stack, creating the situation in Figure 6.9. Next the two operations

Attach to DET

Attach to HEAD

would successfully build the NP from the lexical entries for "*the*" and "*fish*". The input buffer would now be empty.

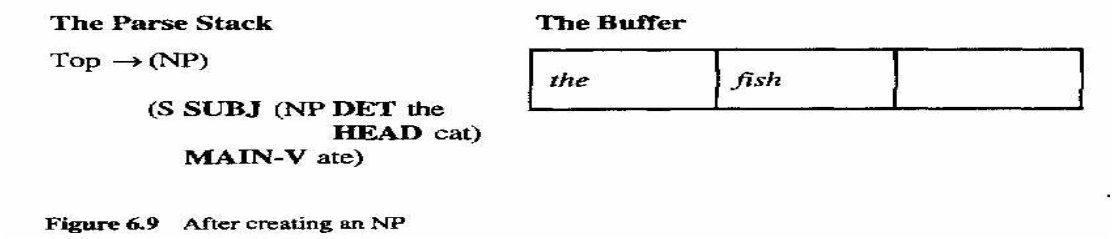


Figure 6.9 After creating an NP

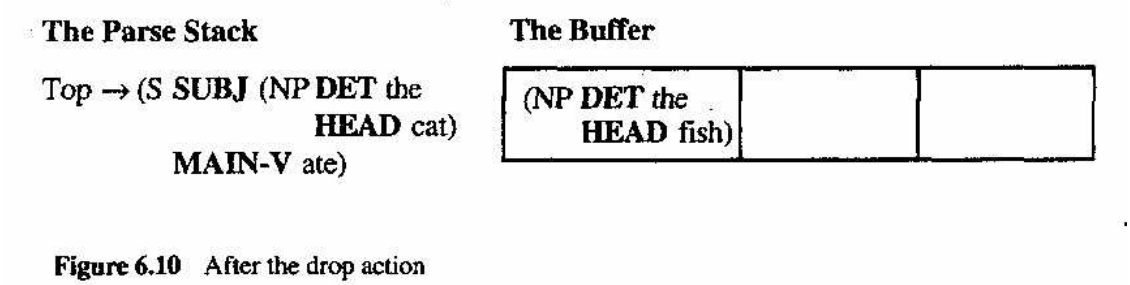


Figure 6.10 After the drop action

The operation

Drop

pops the NP from the parse stack and pushes it back onto the buffer, creating the situation in Figure 6.10.

PBR VITS (AUTONOMOUS)

IV B.TECH CSE-IOT

NATURAL LANGUAGE PROCESSING

TEXT BOOK: James Allen, **Natural Language Understanding**, 2nd Edition, 2003, Pearson Education

Course Instructor: Dr KV Subbaiah, M.Tech, Ph.D, Professor, Dept. of CSE

UNIT–IV Interpretation and Modelling

Semantic Interpretation-Semantic & Logical form, Word senses & ambiguity, the basic logical form language, encoding ambiguity in the logical Form, Verbs & States in logical form, Thematic roles, Speech acts &embedded sentences, Defining semantics structure model theory.
Language Modelling-Introduction, n-Gram Models, Language model Evaluation, Parameter Estimation, Language Model Adaption, Types of Language Models, Language-Specific Modelling Problems, Multilingual and Cross lingual Language Modelling.

4.1 Semantic Interpretation-Semantic & Logical form

Precisely defining the notions of semantics and meaning is surprisingly difficult because the terms are used for several different purposes in natural and technical usage. For instance, there is a use of the verb "*mean"* that has nothing to do with language. Say you are walking in the woods and come across a campfire that is just noticeably warm. You might say "*This fire means someone camped here last night"*. By this you mean that the fire is evidence for the conclusion or implies the conclusion. This is related to, but different from, the notion of meaning that will be the focus of the next few chapters. The meaning we want is closer to the usage when defining a word, such as in the sentence *"'Amble' means to walk slowly"*. This defines the meaning of a word in terms of other words. To make this more precise, we will have to develop a more formally specified language in which we can specify meaning without having to refer back to natural language itself. But even if we can do this, defining a notion of sentence meaning is difficult. For example, I was at an airport recently and while I was walking towards my departure gate, a guard at the entrance asked, "Do you know what gate you are going to?" I interpreted this as asking whether I knew where I was going and answered yes. But this response was based on a misunderstanding of what the guard meant, as he then asked, "Which gate is it?" He clearly had wanted me to tell him the gate number. Thus the sentence "*Do you know what gate you are going to?"* appears to mean different things in different contexts.

Can we define a notion of sentence meaning that is independent of context? In other words, is there a level at which the sentence "*Do you know what gate you are going to?"* has a single meaning, but may be used for different purposes? This is a complex issue, but there are many advantages to trying to make such an approach work. The primary argument is modularity. If such a division can be made, then we can study sentence meaning in detail without all the complications of sentence usage. In particular, if sentences have no context-independent meaning, then we may not be able to separate the study of language from the study of general human reasoning and context. As you will see in the next few chapters, there are many examples of constraints based on the meaning of words that appear to be independent of context. So from now on, we will use the term "*meaning"* in this context-independent sense, and we will use the term "*usage"* for the

context-dependent aspects. The representation of context-independent meaning is called the logical form. The process of mapping a sentence to its logical form is called semantic interpretation, and the process of mapping the logical form to the final knowledge representation (KR) language is called contextual interpretation. Figure 8.1 shows a simple version of the stages of interpretation. The exact meaning of the notations used will be defined later.

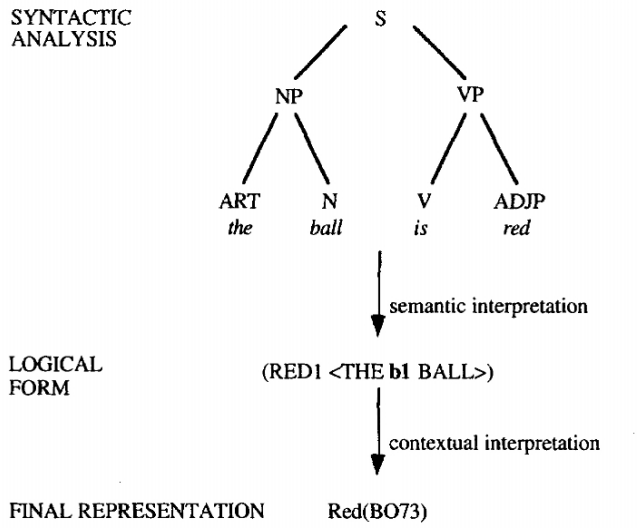


Figure 8.1 Logical form as an intermediate representation

For the moment let us assume the knowledge representation language is the first-order predicate calculus (FOPC). Given that assumption, what is the status of the logical form? In some approaches the logical form is defined as the literal meaning of the utterance, and the logical form language is the same as the final knowledge representation language. If this is to be a viable approach in the long run, however, it would mean that the knowledge representation must be considerably more complex than representations in present use in Al systems. For instance, the logical form language must allow indexical terms, that is, terms that are defined by context. The pronouns "*I"* and "*you"* are indexical because their interpretation depends on the context of who is speaking and listening. In fact most definite descriptions (such as "*the red ball")* are indexical, as the object referred to can only be identified with respect to a context. Many other aspects of language, including the interpretation of tense and determining the scope of quantifiers, depend on context as well and thus cannot be uniquely determined at the logical form level. Of course, all of this could be treated as ambiguity at the logical form level, but this would be impractical, as every sentence would have large numbers of possible logical forms (as in the sentence "*The red ball dropped"*, which would have a different logical form for every possible object that could be described as a ball that is red).

But if the logical form language is not part of the knowledge representation language, what is its formal status? A promising approach has been developed in linguistics over the last decade that suggests an answer that uses the intuitive notion of the meaning of "*situation"* in English. For instance, when attending a class, you are in a situation where there are fellow students and an instructor, where certain utterances are made by the lecturer, questions asked, and so on. Also, there will be objects in the lecture hail, say a blackboard and chairs, and so on. More formally, you might think of a situation as a set of objects and relations between those objects. A very simple situation might consist of two objects, a ball B0005 and a person P86,

and include the relationship that the person owns the ball. Let us encode this situation as the set {(BALL B0005), (PERSON P86), (OWNS P86B0005)}.

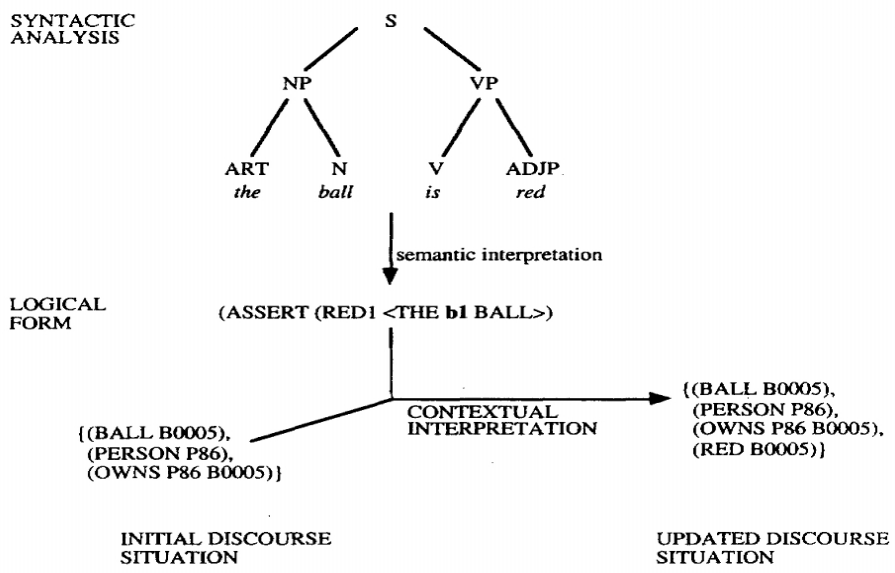


Figure 8.2 Logical form as a function

Language creates special types of situations based on what information is conveyed. These issues will be explored in detail later, but for now consider the following to help your intuition. In any conversation or text, assume there is a discourse situation that records the information conveyed so far. A new sentence is interpreted with respect to this situation and produces a new situation that includes the information conveyed by the new sentence. Given this view, the logical form is a function that maps the discourse situation in which the utterance was made to a new discourse situation that results from the occurrence of the utterance. For example, assume that the situation we just encoded has been created by some preceding sentences describing the ball and who owns it. The utterance "*The ball is red"* might produce a new situation that consists of the old situation plus the new fact that B0005 has the property RED: ((BALL B0005), (PERSON P86), (OWNS P86 B0005*)*, (RED B0005)}. Figure 8.2 shows this view of the interpretation process, treating the logical form as a function between situations. The two organizations presented in Figures 8.1 and 8.2 differ in that the latter might not include a single identifiable expression in the knowledge representation that fully captures the ”meaning” of the sentence. Rather, the logical form might make a variety of changes to produce the updated situation. This allows other implications to be derived from an utterance that are not directly captured in the semantic content of the sentence. Such issues will become important later when we discuss contextual interpretation.

4.2 Word senses & ambiguity

To develop a theory of semantics and semantic interpretation, we need to develop a structural model, just as we did for syntax. With syntax we first introduced the notion of the basic syntactic classes and then developed ways to constrain how simple classes combine to form larger structures. We will follow the same basic strategy for semantics. You might think that the basic semantic unit could be the word or the morpheme, but that approach runs into problems because of the presence of ambiguity. For example, it is

not unusual for the verb *go* to have more than 40 entries in a typical dictionary. Each one of these definitions reflects a different sense of the word.
Dictionaries often give synonyms for particular word senses. For *go* you might find synonyms such as *move, depart, pass, vanish, reach, extend,* and *set out.* Many of these highlight a different sense of the verb *go.* Of course, if these are true synonyms of some sense of *go,* then the verbs themselves will share identical senses. For instance, one of the senses of *go* will be identical to one of the senses of *depart.*

If every word has one or more senses then you are looking at a very large number of senses, even given that some words have synonymous senses. Fortunately, the different senses can be organized into a set of broad classes of objects by which we classify the world. The set of different classes of objects in a representation is called its ontology. To handle a natural language, we need a much broader ontology than commonly found in work on formal logic. Such classifications of objects have been of interest for a very long time and arise in the writings of Aristotle (384—322 B.C.). The major classes that Aristotle suggested were substance (physical objects), quantity (such as numbers), quality (such as bright red), relation, place, time, position, state, action, and affection. To this list we might add other classes such as events, ideas, concepts, and plans. Two of the most influential classes are actions and events. Events are things that happen in the world and are important in many semantic theories because they provide a structure for organizing the interpretation of sentences. Actions are things that agents do, thus causing some event. Like all objects in the ontology, actions and events can be referred to by pronouns, as in the discourse fragment

We lifted the box. It was hard work.

Here, the pronoun "*it"* refers to the action of lifting the box. Another very influential category is the situation. As previously mentioned, a situation refers to some particular set of circumstances and can be viewed as subsuming the notion of events. In many cases a situation may act like an abstraction of the world over some location and time. For example, the sentence "*We laughed and sang at the football game"* describes a set of activities performed at a particular time and location, described as the situation "*the football game"*. Not surprisingly, ambiguity is a serious problem during semantic interpretation. We can define a word as being semantically ambiguous if it maps to more than one sense. But this is more complex than it might first seem, because we need to have a way to determine what the allowable senses are. For example,

4.3 The Basic Logical Form Language

The basic logical form language defines a language in which you can combine word senses elements to form meanings for more complex expressions. This language will follow **First-Order Predicative Calculus** (FOPC). The FOPC are many equivalent forms of representation, such as network-based representations, that use the same basic ideas. The word senses will serve as the atoms or constants of the representation. These constants can be classified by the types of things they describe. For instance, constants that describe objects in the world,

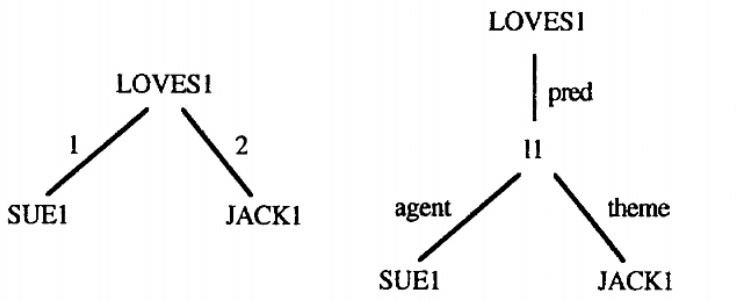


Figure 8.3 Two possible network representations of *Sue loves Jack*

including abstract objects such as events and situations, are called terms. Constants that describe relations and properties are called predicates. A proposition in the language is formed from a predicate followed by an appropriate number of terms to serve as its arguments. For instance, the proposition corresponding to the sentence "*Fido is a dog"* would be constructed from the term FIDO1 and the predicate constant DOG1 and is written as

(DOG1 FIDO1)

Predicates that take a single argument are called unary predicates or properties; those that take two arguments, such as LOVES1, are called binary predicates; and those that take *n* arguments are called n-ary predicates. The proposition corresponding to the sentence *Sue loves Jack* would involve a binary predicate LOVES1 and would be written as

(LOVES1 SUE1 JACK1)

You can see that different word classes in English correspond to different types of constants in the logical form. Proper names, such as *Jack,* have word senses that are terms; common nouns, such as *dog,* have word senses that are unary predicates; and verbs, such as *run, love,* and *put,* have word senses that correspond to *n-ary* predicates, where *n* depends on how many terms the verb subcategorizes for.

Note that while the logical forms are presented in a predicate-argument form here, the same distinctions are made in most other meaning representations. For instance, a network representation would have nodes that correspond to the word senses and arcs that indicate the predicate-argument structure. The meaning of the sentence *Sue loves Jack* in a semantic network like representation might appear in one of the two forms shown in Figure 8.3. For most purposes all of these representation formalisms are equivalent.

More complex propositions are constructed using a new class of constants called logical operators. For example, the operator NOT allows you to construct a proposition that says that some proposition is not true. The proposition corresponding to the sentence *Sue does not love Jack* would be

(NOT (LOVES1 SUE1 JACK1))

English also contains operators that combine two or more propositions to form a complex proposition. FOPC contains operators such as disjunction (v), conjunction (&), what is often called implication **(`),** and other forms (there are 16 possible truth functional binary operators in FOPC). English contains many similar operators including *or, and, if only if,* and so on. Natural language connectives often involve more complex relationships between sentences. For instance, the conjunction "*and"* might correspond to the logical operator "&" but often also involves temporal sequencing, as in "*I went home and had a drink",* in which going home preceded having the drink. The connective "*but",* on the other hand, is like "*and"* except that the second argument is something that the hearer might not expect to be true given the first argument.

The general form for such a proposition is *(connective proposition proposition).* For example, the logical form of the sentence "*Jack loves Sue or Jack loves Mary"* would be **(OR1 (LOVES1 JACK1 SUE1) (LOVES1 JACK1 MARY1))**. The logical form language will allow both operators corresponding to word senses and operators like "&" directly from FOPC. The logic based operators will be used to connect propositions not explicitly conjoined in the sentence.

4.4 Encoding Ambiguity in the Logical Form

The basic logical form language defines a language in which you can combine word senses elements to form meanings for more complex expressions. The Logical form language will follow **First-Order Predicative Calculus** (FOPC) to represent the given corpus of the NLP.

The Need for Generalized Quantifiers:

The Quantifier is used to present more complex sentences by using FOPC and there are only two quantifiers: **Ž** and **j**. English contains a much larger range of quantifiers, including *all, some, most, many, a few, the,* and so on. Thus two instances of the same variable x occurring in two different formulas - say in the formulas **jx.P(x)** and **jx. Q(x).**

For the standard existential and universal quantifiers, there are formulas in standard FOPC equivalent to the generalized quantifier forms. In particular, the formula

(EXISTS x : Px Qx)
is equivalent to
j x . Px & Qx)

and the universally quantified form
(ALL x : Px Qx)
is equivalent to
Žx . Px ` Qx)

These generalized quantifier forms can be thought of simply as abbreviations. But the other quantifiers do not have an equivalent form in standard FOPC. To see this, consider trying to define the meaning of "*Most dogs bark"* using the standard semantics for quantifiers. Clearly **Žx . Dog(x) ` Bark(x)**is too strong, and **jx . Dog(x) ` Bark(x)** is too weak.
Thus the sentence "*Sue watched the ball"* is ambiguous out of context. A single logical form can represent these two possibilities, however:
1. (THE **b1** : ({BALL1 BALL2} **b1**) (PAST (WATCH1 SUE1 **b1**)))
This abbreviates two possible logical forms, namely
2. (THE **b1** : (BALL1 **b1**) (PAST (WATCH1 SUE1 **b1**)))
and
3. (THE **b1** : (BALL2 **b1**) (PAST (WATCH1 SUE1 **b1**)))

Encoding Ambiguity

A typical sentence will have multiple possible syntactic structures, each of which might have multiple possible logical forms.
The words in the sentence will have multiple senses. Many researchers view this ambiguity encoding as a separate level of representation from the logical form, and it is often referred to as the **quasi-logical form**.

For example, the logical forms for the sentence "Every *boy loves a dog"* are captured by a single ambiguous form
(LOVES1 <EVERY **b1** (BOY1 **b1**)> <A **d1** (DOG1 **d1**)>)
This abbreviates an ambiguity between the logical form
(EVERY **b1** : (BOY1 **b1**) (A **d1** (DOG1 **d1**) (LOVES1 **b1 d1**)))
and
(A **d1**: (DOG1 **d1**) (EVERY **b1** : (BOY1 **b1**) (LOVES1 **b1 d1**)))

In addition, operators such as negation and tense are also scope sensitive. For example, the sentence "*Every boy didn't run"* is ambiguous between the read ing in which some boys didn't run and some did, that is,
(NOT (EVERY **b1** : (BOY1 **b1**) (RUN1 **b1**)))

In fact, proper names must be interpreted in context, and the name *John* will refer to different people in different situations. We will introduce this construct as a special function, namely
(NAME <variable> <name>)
which produces the appropriate object with the name in the current context. Thus, the logical form of *"John ran"* would be **(<PAST RUN1> (NAME j1 "John")).**

4.5 Verbs & States in Logical Form

The verbs have mapped to appropriate senses acting as predicates in the logical form. This treatment can handle all the different forms but loses some generalities that could be captured. It also has some annoying properties. Consider the following sentences, all using the verb "*break":*

John broke the window with the hammer.
The hammer broke the window.
The window broke.

The verb "*break"* mapped to the same sense in each case.
The first seems to be a ternary relation between John, the window, and the hammer, the second a binary relation between the hammer and the window, and the third a unary relation involving the window. It seems you would need three different senses of break, BREAK1, BREAK2, and BREAK3, that differ in their arity and produce logical forms such as

1. **(<PAST BREAK1> (NAME j1 "John") <THE w1 WINDOW1> <THE h1 HAMMER1>),**
2. **(<PAST BREAK2> <THE h1 HAMMER1> <THE w1 WINDOW1>), and**
3. **(<PAST BREAK3> <THE w1 WINDOW1>)**

Furthermore, to guarantee that each predicate is interpreted appropriately, the representation would need axioms so that whenever 1 is true, 2 is true, and whenever 2 is true, 3 is true. These axioms are often called meaning postulates.

In particular, the quasi-logical form for the sentence "*John broke the window"* using this abbreviation is
(<PAST BREAK!> **e1** [AGENT (NAME **j1** "John")]
[THEME <THE **w1** WINDOW1>])
It turns out that similar arguments can be made for verbs other than event verbs. Consider the sentence "*Mary was unhappy"*. If it is represented using a unary predicate as
(<PAST UNHAPPY> (NAME **j1** "Mary"))

In many situations using explicit event and state variables in formulas is cumbersome and interferes with the development of other ideas. As a result, we will use different representations depending on what is best for the presentation. For example, the logical form of ***"Mary sees John"*** will sometimes be written as
(PRES (SEES1 **l1** [AGENT (NAME **j1** "Mary")]
[THEME (NAME **m1** "John")]))
which of course is equivalent to
(PRES (j **l1** (& (SEES1 **l1**) (AGENT **l1** (NAME **j1** "Mary"))
(THEME **l1** (NAME **m1** "John")))))

4.6 Thematic roles

Thematic Roles and theories based on the notion of thematic roles, or cases. One of the motivating example is given below.

- **John broke the window with the hammer.**
- **The hammer broke the window.**
- **The window broke.**

John", " the hammer", and "*the window"* play the same semantic roles in each of these sentences. "*John"* is the actor, "*the window"* is the object, and "*the hammer"* is the instrument used in the act of breaking of the window.

We introduced relations such as AGENT, THEME, and INSTR to capture these intuitions.
Perhaps the easiest thematic role to define is the AGENT role.
A noun phrase fills the AGENT role if it describes the instigator of the action described by the sentence. The following sentences are acceptable: (bcz Agent role)

- **John intentionally broke the window.**
- **John broke the window in order to let in some air.**

But these sentences are not acceptable:
*** The hammer intentionally broke the window.**
*** The window broke in order to let in some air.**

For example, given the sentence "***The gray eagle saw the mouse***", the NP "***the mouse***" is the THEME and is the answer to the question "What was seen?"
For intransitive verbs, the THEME role is used for the subject NPs that are not AGENTs. Thus in "***The clouds appeared over the horizon***", the NP "*the clouds*" fills the THEME role. More examples follow, with the THEME NP in italics:

- ***The rock*** broke.
- John broke ***the rock***.
- I gave John ***the book***.

Other phrases describe changes in location, direction of motion, or paths:

- I walked ***from here to school*** yesterday.
- It fell ***to the ground***.
- The birds flew ***from the lake along the river gorge***.

There are at least three different types of phrases here: those that describe where something came from (the FROM-LOC role), such as "*from here";* those that describe the destination (the TO-LOC role), such as "*to the ground";* and those that describe the trajectory or path (the PATH-LOC role), such as "*along the gorge*".

You can see other specializations of these roles when you consider the abstract relation of possession:

- **I threw the ball *to John.* (the TO-LOC role)**
- **I gave a book *to John.* (the TO-POSS role)**
- **I caught the ball *from John.* (the FROM-LOC role)**
- **I borrowed a book *from John.* (the FROM-POSS role)**
- ***The box* contains a ball. (the AT LOC role)**
- ***John* owns a book. (the AT POSS role)**

Similarly, you might define AT-TIME, TO-TIME, and FROM-TIME roles, as in

- **I saw the car *at 3 o'clock.* (the AT-TIME role)**
- **I worked *from one until three.* (the FROM-TIME and TO-TIME role)**
- **The temperature remains *at zero.* (AT VALUE)**
  The temperature rose *from zero.* (FROM-VALUE)

4.7 Speech Acts and Embedded Sentences

Sentences are used for many different purposes. Each sentential mood indicates a different relation between the speaker and the propositional content of the context. The logical form language is extended to

capture the distinctions. Each of the major sentence types has a corresponding operator that takes the sentence interpretation as an argument is called a surface speech act.

They are indicated by new operators as follows:

- **ASSERT** - the proposition is being asserted.
- **Y/N-QUERY** - the proposition is being queried.
- **COMMAND** - the proposition describes an action to perform.
- **WH-QUERY** - the proposition describes an object to be identified.

For ASSERT declarative sentences, such as "***The man ate a peach***", the complete LF is (logical form)

- (ASSERT (<PAST EAT> **e1** [AGENT <THE **m1** MAN1] [THEME <A **p1** PEACH1>]))

For YES/NO questions, such as "*Did the man eat a peach?*", the LF is

- (Y/N-QUERY (<PAST EAT> **e1** [AGENT <THE **m1** MAN1>] [THEME <A **p1** PEACH1>]))

For COMMAND, such as "*Eat the peach*", the LF is

- (COMMAND (EAT **e1** [THEME <THE **p1** PEACH1>]))

For WH-QUERY, the logical form of the sentence "*What did the man eat?*" is

- (WH-QUERY (<PAST EAT> **e1** (AGENT <THE **m1** MAN1>] [THEME <WH **w1** PHYSOBJ>]))

Embedded sentences

Embedded sentences, such as relative clauses, end up as complex restrictions within the noun phrase construction and thus do not need any new notation. For example, the logical form of the sentence "*The man who ate a peach left*" would be

(ASSERT
(<PAST LEAVE> **l1**
[AGENT <THE **m1** (& (MAN1 **m1**)
(<PAST EAT1> **e2** [AGENT **m1**]
[THEME <A **p1** PEACH>]))>]))

4.8 Defining Semantic Structure: Model Theory

The basic building block for defining semantic properties is known as a model. A model theory can be thought of as a set of objects and their properties and relationships, together with a specification of how the language being studied relates to those objects and relationships.

A model can be thought of as representing a particular context in which a sentence is to be evaluated.

For instance, the standard models for logic, called Tarskian models, are complete in that they must map every legal term in the language into the domain and assign every statement to be true or false. Model theory is an excellent method for studying context-independent meaning, because the meanings of sentences are not defined with respect to one specific model but rather by how they relate to any possible model.

Formally, a model m is a tuple **<Dm, Im>,** where **Dm** is the domain of interpretation (that is, a set of primitive objects), and **I** is the interpretation function.

- To handle natural language, the domain of interpretation would have to allow objects of all the different types of things that can be referred to, including physical objects, times, Locations, events, and situations.
- The interpretation function maps senses and larger structures into structures defined on the domain. For example, the following describe how an interpretation function will interpret the senses based on some lexical classes:

Senses of noun phrases - refer to specific objects; the interpretation function maps each to an element of **Dm**.
Senses of singular common nouns (such as "*dog", "idea", "party")* - identify classes of objects in the domain; the interpretation function maps them to sets of elements from **Dm** (that is, subsets of **Dm**).
Senses of verbs - identify sets of n-ary relations between objects in **D**. The arity depends on the verb.

UTTERANCE → (ASSERT *PROPOSITION*) |
(Y/N-QUERY *PROPOSITION*) |
(COMMAND *PROPOSITION*) |
(WH-QUERY *PROPOSITION*)

PROPOSITION → (*n-ARY-OPERATOR PROPOSITION*$_1$ ... *PROPOSITION*$_n$) |
(*QUANTIFIER VARIABLE : PROPOSITION PROPOSITION*) |
(*n-ARY-PREDICATE TERM*$_1$ ... *TERM*$_n$) |
(*EVENT-STATE-PRED VARIABLE* [*ROLE-NAME TERM*]$_1$ ...
[*ROLE-NAME TERM*]$_n$)

TERM → *VARIABLE* |
(NAME *VARIABLE NAME-STRING*) |
(PRO *VARIABLE PROPOSITION*)

1-ARY-OPERATOR → NOT | PAST | PERF | PROG | ...

2-ARY-OPERATOR → AND | BUT | IF-THEN | ...

QUANTIFIER → THE | SOME | WH | ∃ | ..

VARIABLE → **b1** | **man3** | ...

1-ARY-PREDICATE → *TYPE-PREDICATE* | HAPPY1 | RED1 | ...

TYPE-PREDICATE → *EVENT-STATE-PRED* | (PLUR *TYPE-PREDICATE*) | MAN1 | ...

EVENT-STATE-PRED → RUN1 | LOVE3 | GIVE1 | HAPPY | ...

2-ARY-PREDICATE → *ROLE-NAME* | ABOVE1 | ...

ROLE-NAME → AGENT | THEME | AT-LOC | INSTR | ...

NAME-STRING → "John" | "The New York Times" | ...

Figure 8.7 A formal definition of the syntax of the logical form language

$TERM \rightarrow \langle QUANTIFIER\ VARIABLE\ PROPOSITION \rangle$

$TERM \rightarrow \langle n\text{-}ARY\text{-}OPERATOR\ TERM_1\ \ldots\ TERM_n \rangle$

$n\text{-}ARY\text{-}PREDICATE \rightarrow \langle m\text{-}ARY\text{-}OPERATOR\ n\text{-}ARY\text{-}PREDICATE_1 \ldots n\text{-}ARY\text{-}PREDICATE_m \rangle$

$n\text{-}ARY\text{-}OPERATOR \rightarrow \{n\text{-}ARY\text{-}OPERATOR_1 \ldots n\text{-}ARY\text{-}OPERATOR_m\}$

$QUANTIFIER \rightarrow \{QUANTIFIER_1 \ldots QUANTIFIER_m\}$

$n\text{-}ARY\text{-}PREDICATE \rightarrow \{n\text{-}ARY\text{-}PREDICATE_1 \ldots n\text{-}ARY\text{-}PREDICATE_m\}$

$TYPE\text{-}PREDICATE \rightarrow \{TYPE\text{-}PREDICATE_1 \ldots TYPE\text{-}PREDICATE_m\}$

$EVENT\text{-}STATE\text{-}PRED \rightarrow \{EVENT\text{-}STATE\text{-}PRED_1 \ldots EVENT\text{-}STATE\text{-}PRED_m\}$

$ROLE\text{-}NAME \rightarrow \{ROLE\text{-}NAME_1 \ldots ROLE\text{-}NAME_m\}$

Figure 8.8 Additional rules defining the quasi-logical form

4.9 Language Modelling-Introduction and n-Gram Models

The Models that assign probabilities to sequences of words are called **language models (LMs).** The simplest language model that assigns probabilities to sentences and sequences of words is the **n-gram** model.

An **n-gram**is a sequence of N words: For example "**Please turn your homework**" can be written as:
–A 1-gram (unigram) is a single word sequence of words like "please", "turn", "your", "homework"

–A 2-gram (bigram) is a two-word sequence of words like "please turn", "turn your", or "your homework".

–A 3-gram (trigram) is a three-word sequence of words like "please turn your", or "turn your homework".

We can use n-gram models to estimate the probability of the last word of an n-gram given the previous words, and also to assign probabilities to entire word sequences.

Probabilistic Language Models:

Probabilistic language models can be used to assign a probability to a sentence in many NLP tasks.

•Machine Translation: P(highwinds tonight) > P(largewinds tonight)
•Spell Correction: Thek office is about ten minutes from here
P(The Office is) > P(Then office is)

•Speech Recognition:

P(I saw a van) >> P(eyes awe of an)

•Summarization, question-answering.

Our goal is to compute the probability of a sentence or sequence of words W (=w1,w2,…wn):

P(W) = P(w1,w2,w3,w4,w5…wn)

•What is the probability of an upcoming word?: P(w5|w1,w2,w3,w4)

•A model that computes either of these: P(W) or P(wn|w1,w2…wn-1) is called a **language model**.

Chain Rule of Probability

We compute probabilities of entire word sequences like w1,w2,…wn

The probability of the word sequence w1,w2,…wn is P(w1,w2,…wn).

We can use the **chain rule of the probability** to decompose this probability:

$$P(w_1^n) = P(w_1)\,P(w_2|w_1)\,P(w_3|w_1^2)\,\ldots\,P(w_n|w_1^{n-1})$$

$$= \prod_{k=1}^{n} P(w_k \mid w_1^{k-1})$$

Example: P(the man from jupiter) =

P(the) P(man|the) P(from|the man) P(jupiter|the man from)

N-Grams

The aim of the n-gram model (simplifying assumption) is to predict the next word in the given text. Instead of computing the probability of a word given its entire history, we can approximate the history by just the last few words.

$P(w_n\|w_1\ldots w_{n-1}) \approx P(w_n)$	**unigram**
$P(w_n\|w_1\ldots w_{n-1}) \approx P(w_n\|w_{n-1})$	**bigram**
$P(w_n\|w_1\ldots w_{n-1}) \approx P(w_n\|w_{n-1}w_{n-2})$	**trigram**
$P(w_n\|w_1\ldots w_{n-1}) \approx P(w_n\|w_{n-1}w_{n-2}w_{n-3})$	**4-gram**
$P(w_n\|w_1\ldots w_{n-1}) \approx P(w_n\|w_{n-1}w_{n-2}w_{n-3}w_{n-4})$	**5-gram**

- In general, **N-Gram** is

$$P(w_n|w_1\ldots w_{n-1}) \approx P(w_n|w_{n-N+1}^{n-1})$$

N-Grams

Computing probabilities of word sequences (Sentences)

Unigram

P(<s> the man from Jupiter came </s>)
P(the) P(man) P(from) P(jupiter) P(came)

Bigram
P(<s> the man from Jupiter came </s>)
P(the|<*s*>) P(man|the) P(from|man) P(jupiter|from) P(came|jupiter) P(</s>|came)

Trigram
P(<s> the man from Jupiter came </s>)
P(the|<*s*><*s*>) P(man|<*s*>the) P(from|the man) P(jupiter|man from) P(came|from jupiter) P(</s>|Jupiter came) P(</s>|came </s>)

N-Grams and Markov Models

The assumption that the probability of a word depends only on the previous word(s) is called **Markov assumption**.

Markov models are the class of probabilistic models that assume that we can predict the probability of some future unit without looking too far into the past.

•A **bigram** is called a **first-order** Markov model (because it looks one token into the past)
•A **trigram** is called a **second-order** Markov model;
•In general a **N-Gram** is called a **N-1 order** Markov model.

Estimating N-Gram Probabilities

Estimating n-gram probabilities is called **maximum likelihood estimation (**or **MLE**).
We get the MLE estimate for the parameters of an n-gram model by ***getting counts from a corpus***, and **normalizing** the counts so that they lie between 0 and 1.

•**Estimating bigram probabilities:**

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)} = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

where C is the count of that pattern in the corpus

- Estimating N-Gram probabilities

$$P(w_n \mid w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})}$$

Estimating N-Gram Probabilities

A Bigram Example

A mini-corpus: We augment each sentence with a special symbol <s> at the beginning of the sentence, to give us the bigram context of the first word, and special end-symbol </s>.
<s> I am Sam </s>
<s> Sam I am </s>
<s> I fly </s>

Unique words: I, am, Sam, fly

Bigrams: <s> and </s> are also tokens. There are 6(4+2) tokens and 6*6=36 bigrams

P(I|<s>)=2/3	P(Sam|<s>)=1/3	P(am|<s>)=0	P(fly|<s>)=0	P(<s>|<s>)=0	P(</s>|<s>)=0
P(I|I)=0	P(Sam|I)=0	P(am|I)=2/3	P(fly|I)=1/3	P(<s>|I)=0	P(</s>|I)=0
P(I|am)=0	P(Sam|am)=1/2	P(am|am)=0	P(fly|am)=0	P(<s>|am)=0	P(</s>|am)=1/2
P(I|Sam)=1/2	P(Sam|Sam)=0	P(am|Sam)=0	P(fly|Sam)=0	P(<s>|Sam)=0	P(</s>|Sam)=1/2
P(I|fly)=0	P(Sam|fly)=0	P(am|fly)=0	P(fly|fly)=0	P(<s>|fly)=0	P(</s>|fly)=1
P(I|</s>)=0	P(Sam|</s>)=1/3	P(am|</s>)=1/3	P(fly|</s>)=1/3	P(<s>|</s>)=0	P(</s>|</s>)=0

Estimating N-Gram Probabilities

Example

<s> I am Sam </s>
<s> Sam I am </s>
<s> I fly </s>

Unigrams: I, am, Sam, fly

P(I)=3/8 P(am)=2/8 P(Sam)=2/8 P(fly)=1/8

Trigrams: There are 6*6*6=216 trigrams.

– Assume there are two tokens <s> <s> at the begining, and two tokens </s> </s> at the end.

P(I|<s> <s>)=2/3 P(Sam|<s> <s>)=1/3
P(am|<s> I)=1/2 P(fly|<s> I)=1/2
P(I|<s> Sam)=1
P(Sam|I am)=1/2 P(</s>|I am)=1/2
P(</s>|am Sam)=1
P(</s>|Sam </s>)=1

4.10 Language model Evaluation

Language model evaluation in Natural Language Processing (NLP) is a crucial step to assess the performance and quality of a language model. Evaluating language models helps researchers and developers understand how well a model performs on various tasks and benchmark it against other models or baselines. Here are some common evaluation techniques used in NLP:

1. **Perplexity:** Perplexity is a widely used metric for evaluating language models. It measures how well a language model predicts a given text corpus. Lower perplexity indicates better performance. Perplexity is calculated using the probabilities assigned by the model to each word in a test set.
2. **Accuracy:** Accuracy is a common evaluation metric for tasks like sentiment analysis, text classification, or named entity recognition. It measures the proportion of correctly predicted instances compared to the total number of instances in a test set. Accuracy is usually calculated by comparing the predicted labels or outputs of the model with the ground truth labels.
3. **F1 Score:** F1 score is a widely used metric for evaluating models in tasks like named entity recognition, part-of-speech tagging, and information extraction. It balances precision (the proportion of correctly predicted

positive instances) and recall (the proportion of actual positive instances correctly predicted). F1 score is the harmonic mean of precision and recall and provides a single measure of model performance.

4. **BLEU Score:** BLEU (Bilingual Evaluation Understudy) is a metric commonly used in machine translation to evaluate the quality of generated translations. It measures the overlap between the model's predicted translations and reference translations. BLEU score ranges from 0 to 1, with higher scores indicating better translation quality.
5. **ROUGE Score:** ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a family of metrics used to evaluate text summarization systems. ROUGE measures the overlap between the model's generated summaries and reference summaries. Different variants of ROUGE, such as ROUGE-N (measuring n-gram overlap) and ROUGE-L (measuring longest common subsequence), are used to evaluate different aspects of summarization quality.
6. **Human Evaluation:** In addition to automated metrics, human evaluation is often conducted to assess the quality of language models. Human evaluators provide judgments on various aspects of model outputs, such as fluency, coherence, grammaticality, relevance, and overall quality. Human evaluation provides valuable insights that may not be captured by automated metrics.

It is important to note that the choice of evaluation metric depends on the specific NLP task and the goals of the evaluation. Different metrics have their strengths and weaknesses, and a combination of metrics is often used to provide a comprehensive evaluation of a language model's performance.

4.11 Parameter Estimation

Parameter estimation in natural language processing (NLP) refers to the process of estimating the values of model parameters based on observed data. In NLP, parameter estimation is crucial for various tasks, such as language modeling, machine translation, sentiment analysis, and part-of-speech tagging, among others.

Types of Parameter Estimations

1. Maximum-Likelihood Estimation and Smoothing (MLE)
2. Bayesian Parameter Estimation
3. Large-Scale Language Models

1.Maximum-Likelihood Estimation and Smoothing (MLE)

Maximum Likelihood Estimation (MLE) is a common technique used in Natural Language Processing (NLP) to estimate the parameters of a probabilistic language model. MLE aims to find the parameter values that maximize the likelihood of the observed data.

In the context of NLP, MLE is often used to estimate the probabilities of words or sequences of words in a language model. The basic idea is to count the occurrences of different words or sequences of words in a given training corpus and use these counts to calculate the probabilities as follows.

P(wi/wi-1,wi-2)=c(wi, wi-1, wi-2)/c(wi-1,wi-2)

Here's a step-by-step explanation of how MLE works in NLP:

1. **Data Collection:** Collect a large corpus of text that represents the domain or language you want to model. This corpus will be used as the training data.
2. **Data Preprocessing:** Preprocess the training data by tokenizing it into words or subword units, removing punctuation, normalizing case, etc. This step ensures that the data is in a suitable format for modeling.
3. **Parameter Estimation:** Calculate the probabilities of words or sequences of words based on the training data. For example, to estimate the probability of a word, count the number of occurrences of that word in the corpus and divide it by the total number of words in the corpus.
4. **Smoothing:** One challenge in using MLE is that unseen words or sequences in the training data will have zero probabilities. To address this issue, smoothing techniques are often applied to assign non-zero probabilities to unseen events. Common smoothing methods include Laplace smoothing (add-one smoothing), Lidstone smoothing, and Good-Turing smoothing.
5. **Model Evaluation:** Evaluate the performance of the language model using evaluation metrics such as perplexity, accuracy, or F1 score. These metrics help assess how well the model predicts the target language or performs on specific NLP tasks.

MLE is a fundamental approach for estimating language model probabilities in NLP. However, it has some limitations. MLE assumes that the training data is representative of the target distribution and that each data point is independent. In practice, these assumptions may not always hold, leading to potential limitations in the performance of the language model. Researchers and practitioners often explore more sophisticated techniques, such as neural language models, to overcome these limitations and improve language modeling in NLP.

2. Bayesian Parameter Estimation

Bayesian parameter estimation is an alternative approach to estimating the parameters of a language model in Natural Language Processing (NLP). Unlike Maximum Likelihood Estimation (MLE), which provides point estimates of the parameters, Bayesian estimation incorporates prior knowledge and uncertainty into the parameter estimation process.

In Bayesian parameter estimation, the goal is to estimate the posterior distribution of the parameters given the observed data and prior knowledge. The posterior distribution represents the updated belief about the

parameter values after considering the observed data. This distribution is obtained by combining the likelihood of the data and the prior distribution of the parameters using Bayes' theorem.

Here's a step-by-step explanation of how Bayesian parameter estimation works in NLP:

1. Define a Prior Distribution: Specify a prior distribution that captures the initial belief or knowledge about the parameters before observing any data. The prior distribution represents the uncertainty or prior information about the parameter values.
2. Likelihood Calculation: Calculate the likelihood of the observed data given the parameters. The likelihood measures how probable the observed data is under the current parameter values.
3. Bayesian Inference: Apply Bayes' theorem to update the prior distribution based on the likelihood and obtain the posterior distribution of the parameters. The posterior distribution reflects the updated belief about the parameter values after considering the observed data.
4. Posterior Analysis: Analyze the posterior distribution to obtain estimates of the parameters. This can be done by calculating the mean, median, or mode of the posterior distribution, which represent the point estimates of the parameters. Additionally, credible intervals can be computed to quantify the uncertainty in the parameter estimates.
5. Model Evaluation: Evaluate the performance of the Bayesian language model using evaluation metrics such as perplexity, accuracy, or F1 score, similar to the evaluation of MLE-based models.

Bayesian parameter estimation provides several advantages over MLE in NLP. It allows the incorporation of prior knowledge, which can be particularly useful when the amount of available training data is limited. Additionally, Bayesian estimation provides a richer representation of uncertainty by yielding a posterior distribution rather than a point estimate. This uncertainty information can be valuable for decision-making or downstream applications.

However, Bayesian parameter estimation can be computationally demanding, especially for complex models with high-dimensional parameter spaces. Techniques like Markov Chain Monte Carlo (MCMC) or variational inference are commonly used to approximate the posterior distribution when direct computation is infeasible.

Overall, Bayesian parameter estimation offers a principled and flexible framework for estimating parameters in language models, incorporating prior knowledge, and capturing uncertainty in NLP tasks.

4.12 Language Model Adaption

Language model adaptation in NLP refers to the process of fine-tuning a pre-trained language model on a specific task or domain to improve its performance. Language models like GPT-3.5 are trained on a large corpus of diverse text from the internet, which gives them a general understanding of language. However, fine-tuning or adapting them to a specific task or domain can enhance their performance on that particular task.

Here are the general steps involved in language model adaptation:

1. **Dataset Collection:** Gather a dataset that is specific to the target task or domain you want to adapt the language model to. This dataset should be representative of the target task and should include text samples that the language model is likely to encounter during inference.
2. **Dataset Preprocessing:** Preprocess the collected dataset by cleaning the text, removing irrelevant information, and ensuring the dataset is in a format that can be used for training.
3. **Model Architecture:** Decide on the specific architecture you want to use for adaptation. This can be the same architecture as the pre-trained language model or a modified version depending on the task requirements.
4. **Task-Specific Labels:** If your target task involves supervised learning, you will need to annotate or label your dataset with the appropriate task-specific labels. This step is crucial for tasks like sentiment analysis, named entity recognition, or machine translation.
5. **Fine-tuning:** Initialize the pre-trained language model with the weights learned during pre-training, and then fine-tune it on your target dataset. During fine-tuning, the model learns task-specific patterns and representations. The process typically involves minimizing a task-specific loss function using techniques such as backpropagation and gradient descent.
6. **Hyperparameter Tuning:** Experiment with different hyperparameters such as learning rate, batch size, and regularization techniques to optimize the performance of the adapted model. This step helps to find the best set of hyperparameters that work well for the specific task.
7. **Evaluation and Iteration:** Evaluate the performance of the adapted language model on a validation set or using other evaluation metrics specific to your task. Iterate on the fine-tuning process by making changes to the architecture, hyperparameters, or dataset, if necessary, to further improve performance.

It's worth noting that language model adaptation requires a substantial amount of task-specific data to achieve good performance. If only a limited amount of data is available, transfer learning techniques such as few-shot or zero-shot learning can be used to leverage the knowledge learned from the pre-trained language model. Additionally, it's important to strike a balance between task-specific adaptation and preserving the general language understanding of the original pre-trained model.

4.13 Types of Language Models

There are various types of language models used in natural language processing (NLP) that differ in their architecture, training methods, and intended use cases. Here are some commonly used types of language models in NLP:

1. **N-gram Language Models**: N-gram models are simple and widely used language models that predict the probability of a word given its preceding context of n-1 words. They rely on counting and statistical techniques to estimate the probabilities. For example, a trigram model predicts the next word based on the previous two words.
2. **Neural Language Models**: Neural language models leverage deep learning techniques, such as recurrent neural networks (RNNs), long short-term memory (LSTM) networks, and transformers, to capture complex patterns and dependencies in language. These models learn distributed representations of words and generate probabilities based on the context.
3. **Transformer-based Language Models**: Transformer models, such as OpenAI's GPT (Generative Pre-trained Transformer), have gained significant attention in recent years. These models use self-attention mechanisms to capture global dependencies and parallelize computation, making them highly effective for tasks like language generation, machine translation, and question answering.

4. **Contextualized Language Models**: Contextualized language models, such as ELMo (Embeddings from Language Models) and BERT (Bidirectional Encoder Representations from Transformers), generate word representations that are sensitive to the context in which the word appears. These models provide contextual embeddings that are useful for downstream NLP tasks like sentiment analysis, named entity recognition, and text classification.
5. **Encoder-Decoder Models**: Encoder-decoder models, often based on the sequence-to-sequence architecture, are used for tasks like machine translation and text summarization. These models consist of an encoder that processes the input sequence and a decoder that generates the output sequence based on the encoded representation.
6. **Hybrid Models**: Some language models combine multiple approaches or architectures to leverage their individual strengths. For example, ULMFiT (Universal Language Model Fine-tuning) combines features of transformer models with techniques like transfer learning and fine-tuning to improve performance on specific tasks.
7. **Pre-trained Language Models**: Pre-trained language models, such as GPT, BERT, and RoBERTa, are models that are trained on large amounts of data before being fine-tuned for specific tasks. These models capture a broad understanding of language and can be adapted to various downstream tasks with additional training.

These are just a few examples of the types of language models used in NLP. Different models have different strengths and are suited for specific tasks and scenarios. Researchers continue to explore new architectures and techniques to develop more powerful and efficient language models for a wide range of NLP applications.

4.14 Language-Specific Modelling Problems

In natural language processing (NLP), there are several language-specific modeling problems that arise due to the unique characteristics and complexities of different languages. Here are some common language-specific modeling challenges in NLP:

1. **Morphological Variations:** Many languages exhibit rich morphology, where words can have multiple forms depending on factors like tense, number, gender, and case. Modeling morphological variations can be challenging, especially for languages with highly inflected forms, as it requires capturing the intricate relationships between word forms and their meanings.
2. **Word Order and Syntax:** Languages vary in their word order and sentence structure. For example, English follows a subject-verb-object (SVO) order, while languages like Japanese follow a subject-object-verb (SOV) order. Modeling the syntactic structure and understanding the correct word order for different languages is crucial for tasks like parsing, machine translation, and text generation.
3. **Named Entity Recognition (NER):** Named Entity Recognition involves identifying and classifying named entities such as names of people, organizations, locations, and dates in text. Different languages have specific naming conventions and variations, which pose challenges for building language-independent NER systems. Language-specific rules and resources are often required to handle these variations effectively.
4. **Sentiment Analysis:** Sentiment analysis aims to determine the sentiment or opinion expressed in a text. The sentiment lexicons, patterns, and linguistic cues that indicate sentiment can differ across languages. Building accurate sentiment analysis models requires language-specific sentiment resources and training data that capture the nuances of sentiment expression in different languages.

5. **Language Idiosyncrasies:** Each language has its own set of idiosyncrasies, such as idiomatic expressions, proverbs, and cultural references. These language-specific nuances pose challenges for models that aim to understand and generate natural language text. Capturing and representing these idiosyncrasies is crucial for building language models that accurately handle language-specific contexts.
6. **Low-Resource Languages:** Low-resource languages, which have limited amounts of training data and linguistic resources, present unique modeling challenges. Limited resources make it difficult to develop robust models for these languages. Techniques like cross-lingual transfer learning, data augmentation, and unsupervised methods are often employed to address the scarcity of language-specific resources.
7. **Code-Switching and Multilingualism:** Many languages exhibit code-switching, where speakers alternate between two or more languages within a conversation. Modeling code-switching and multilingual text requires techniques that can handle language mixing, language identification, and understanding the context-dependent language usage.

Addressing these language-specific modeling challenges in NLP often involves a combination of linguistic knowledge, domain expertise, and language-specific resources. Researchers and practitioners work on developing techniques that can effectively handle these challenges and improve the performance and applicability of NLP models across different languages.

4.15 Multilingual and Cross lingual Language Modelling.

Multilingual and cross-lingual language modeling in NLP involves developing models that can understand, generate, or process multiple languages. Here's an overview of multilingual and cross-lingual language modeling:

1. **Multilingual Language Modeling:** Multilingual language models are trained to handle multiple languages simultaneously. These models are typically trained on a diverse multilingual corpus and aim to capture shared linguistic properties across languages. By leveraging the similarities between languages, multilingual models can generalize well across different language contexts. They can be used for tasks like text classification, named entity recognition, and sentiment analysis in multiple languages.
2. **Cross-lingual Language Modeling:** Cross-lingual language models focus on bridging the gap between different languages. These models are trained on a source language and then applied to a target language, even if the target language has little or no labeled data. Cross-lingual models enable transfer learning, where knowledge learned from a resource-rich language can be transferred to improve performance in a low-resource language. This approach is particularly useful for tasks like machine translation, cross-lingual document retrieval, and cross-lingual information extraction.
3. **Bilingual and Multilingual Embeddings:** Bilingual and multilingual word embeddings represent words from different languages in a shared vector space. These embeddings capture semantic and syntactic similarities between words in multiple languages. Bilingual embeddings map words from one language to their counterparts in another language, while multilingual embeddings aim to

represent words from multiple languages in a single vector space. These embeddings facilitate cross-lingual tasks by allowing for direct comparisons and knowledge transfer across languages.

4. **Machine Translation:** Machine translation is a classic example of cross-lingual language modeling. Neural machine translation models, such as sequence-to-sequence models, use an encoder-decoder architecture to translate text from one language to another. These models can be trained with parallel corpora containing source-target language pairs, and they learn to map the source language to the target language. Transfer learning techniques and multilingual training can improve the translation performance across multiple languages.
5. **Zero-Shot and Few-Shot Learning:** Zero-shot and few-shot learning techniques enable cross-lingual transfer without requiring labeled data in the target language. In zero-shot learning, a model can generate outputs in a language it has not been explicitly trained on, using only high-level instructions or prompts. Few-shot learning involves training a model on a limited amount of labeled data from the target language, allowing it to perform well on that language. These techniques are useful when resources in the target language are scarce.
6. **Cross-lingual Named Entity Recognition (NER):** Cross-lingual NER involves recognizing named entities in multiple languages. By leveraging multilingual training data and shared representations, cross-lingual NER models can generalize across languages and recognize entities even in low-resource languages. Techniques like cross-lingual transfer learning and alignment methods are used to improve cross-lingual NER performance.

Multilingual and cross-lingual language modeling techniques have contributed significantly to making NLP more accessible and effective across different languages and language pairs. They enable the development of models that can handle diverse linguistic contexts, promote knowledge transfer, and facilitate communication and understanding in a multilingual world.

PBR VITS (AUTONOMOUS)

IV B.TECH CSE-IOT

NATURAL LANGUAGE PROCESSING

TEXT BOOK: James Allen, **Natural Language Understanding**, 2nd Edition, 2003, Pearson Education

Course Instructor: Dr KV Subbaiah, M.Tech, Ph.D, Professor, Dept. of CSE

UNIT–V Machine Translation and Multilingual Information

Machine Translation Survey: Introduction, Problems of Machine Translation, Is Machine Translation Possible, Brief History, Possible Approaches, Current Status. Anusaraka or Language Accessor: Background, Cutting the Gordian Knot, The Problem, Structure of Anusaraka System, User Interface, Linguistic Area, Giving up Agreement in Anusaraka Output, Language Bridges.
Multilingual Information Retrieval - Introduction, Document Pre-processing, Monolingual Information Retrieval, CLIR, MLIR, Evaluation in Information Retrieval, Tools, Software and Resources.
Multilingual Automatic Summarization - Introduction, Approaches to Summarization, Evaluation, How to Build a Summarizer, Competitions and Datasets.

---

1. Introduction:

Natural Language Processing (NLP) has revolutionized the field of machine translation, enabling the development of advanced algorithms and models that can automatically translate text or speech from one language to another. This survey focuses on gathering insights and opinions specifically related to machine translation in the context of NLP.

Machine translation plays a vital role in bridging language barriers and promoting global communication. It has applications in various domains, including multilingual customer support, cross-border business collaborations, and information dissemination on the internet. As NLP techniques continue to advance, machine translation systems have become more accurate and efficient, improving the overall quality of translations.

This survey aims to assess the current state of machine translation in the context of NLP, explore the challenges faced by researchers and practitioners, and identify potential areas for future research and development. By participating in this survey, you

will contribute to the understanding of the current landscape of machine translation in the field of NLP.

Your responses will be treated anonymously, and the survey results will be used for research purposes only. Please provide your insights and opinions based on your knowledge and expertise in the field of NLP and machine translation.

2. Problems of Machine Translation

Machine translation in NLP faces several challenges that impact the quality and accuracy of translations. Some of the prominent problems include:

1. **Ambiguity:** Natural languages often contain ambiguous words, phrases, or sentences that can have multiple interpretations. Machine translation systems may struggle to accurately disambiguate such instances, leading to incorrect translations.
2. **Idiomatic Expressions and Cultural Nuances**: Languages contain idiomatic expressions and cultural nuances that are challenging to translate accurately. Machine translation systems may struggle to capture the intended meaning or may produce literal translations that lack the cultural context, resulting in translations that sound awkward or are incorrect.
3. **Syntax and Grammar:** Translating sentences while preserving the correct syntax and grammar is a complex task. Machine translation systems may produce translations that have grammatical errors, incorrect word order, or lack fluency, making them harder to understand.
4. **Out-of-vocabulary (OOV) Words:** Machine translation systems often encounter words or phrases that are not present in their training data. These out-of-vocabulary words pose a challenge as the system may not have learned their translations, leading to inaccurate or untranslated words in the output.
5. **Domain-specific Terminology:** Different domains have their own specific terminologies, such as technical or medical terms, which might not have direct translations. Machine translation systems may struggle to handle these specialized terms, resulting in inaccurate or inconsistent translations.
6. **Language Pair Discrepancies:** The performance of machine translation systems can vary significantly depending on the language pair being translated. Some language pairs have more training data available, leading to better performance, while others may have limited resources, resulting in poorer translations.
7. **Lack of Context Understanding**: Machine translation systems often lack the ability to understand the broader context of a sentence or document. They may struggle with pronoun resolution, coreference resolution, or understanding context-dependent meanings, leading to translation errors.
8. **Rare or Low-Resource Languages:** Machine translation for rare or low-resource languages poses additional challenges due to limited training data and resources. It is often difficult to achieve high-quality translations for these languages due to the scarcity of linguistic resources and models trained specifically for them.

Addressing these challenges requires ongoing research and development in the field of machine translation in NLP. By tackling these problems, we can improve the accuracy, fluency, and overall quality of machine translation systems.

3. Is Machine Translation Possible

Yes, machine translation is a subfield of Natural Language Processing (NLP). Machine translation involves the use of algorithms and computational methods to automatically translate text or speech from one language to another. It aims to bridge the language barrier and facilitate communication between people who speak different languages.

Machine translation can be achieved through various approaches, including rule-based systems, statistical models, and more recently, neural machine translation (NMT). NMT, powered by deep learning techniques, has shown significant improvements in translation quality compared to earlier approaches.

NLP encompasses a wide range of tasks beyond machine translation, such as language understanding, sentiment analysis, question answering, text summarization, and many others. However, machine translation remains one of the fundamental and widely studied applications within the field of NLP.

4. Current Status

As of my knowledge cutoff in September 2021, machine translation has made significant progress in the field of NLP, particularly with the advent of neural machine translation (NMT) models. NMT has revolutionized the quality of machine translation outputs and has become the dominant approach in recent years.

NMT models employ deep learning techniques, specifically neural networks, to learn the mapping between different languages. These models can handle complex sentence structures, capture context dependencies, and generate more fluent translations compared to previous statistical and rule-based methods.

The most notable NMT architecture is the Transformer model, introduced in 2017. Transformers leverage attention mechanisms to focus on relevant parts of the input sequence during translation, allowing for better long-range dependencies modeling. This architecture has achieved state-of-the-art results on various machine translation benchmarks.

Additionally, large-scale pretraining techniques, such as unsupervised or semi-supervised learning, have been explored to enhance machine translation performance.

By leveraging massive amounts of monolingual data, these approaches can improve translation quality even without parallel corpora.

It's important to note that the field of NLP is rapidly evolving, and new techniques and models may have been developed since my last update. I recommend referring to recent research papers, conferences, and advancements in the field to stay up to date with the current state of machine translation in NLP.

5. Cutting the Gordian Knot

"Cutting the Gordian Knot" is a metaphorical expression that originates from the legend of Alexander the Great. According to the legend, Alexander encountered a complex knot tied by Gordius, the king of Phrygia. The knot was said to be impossible to untie, and it was prophesied that whoever could unravel it would become the ruler of Asia. Instead of attempting to untie the knot, Alexander famously took his sword and sliced through it, "cutting the Gordian Knot" and solving the problem in a bold and unconventional way.

In the context of NLP, "cutting the Gordian Knot" can refer to finding a simple and effective solution to a complex problem or challenge. NLP tasks often involve intricate linguistic nuances, ambiguity, and challenges related to language understanding and generation. Researchers and practitioners in NLP continuously seek innovative approaches and techniques to tackle these challenges.

The expression "cutting the Gordian Knot" in NLP may signify the discovery of a breakthrough technique, a novel algorithm, or an innovative model architecture that simplifies or solves a previously difficult or unsolved problem in natural language processing.

It's worth noting that specific techniques and approaches for "cutting the Gordian Knot" in NLP may vary depending on the particular problem or task at hand. Researchers and practitioners employ a range of methods, including deep learning, neural networks, transfer learning, reinforcement learning, and more, to overcome challenges and improve the performance of NLP systems.

---

6. Structure of Anusaraka System

The name Anusaaraka is derived from Sanskrit word Anusaaran that means “to follow”. In the processing of Anusaaraka output appears in one step followed by the next one. Hence it is named so based on its way of generating the output. Anusaaraka is a translator that accepts English as input and produces output in Telugu/Hindi etc. The sentence is passed through various stages of defragmentation and analysis before the output is generated.

The Anusaaraka architecture has been designed and developed based on issues revealed during an evaluation of conventional machine translation. The architecture is shown in Fig 1.

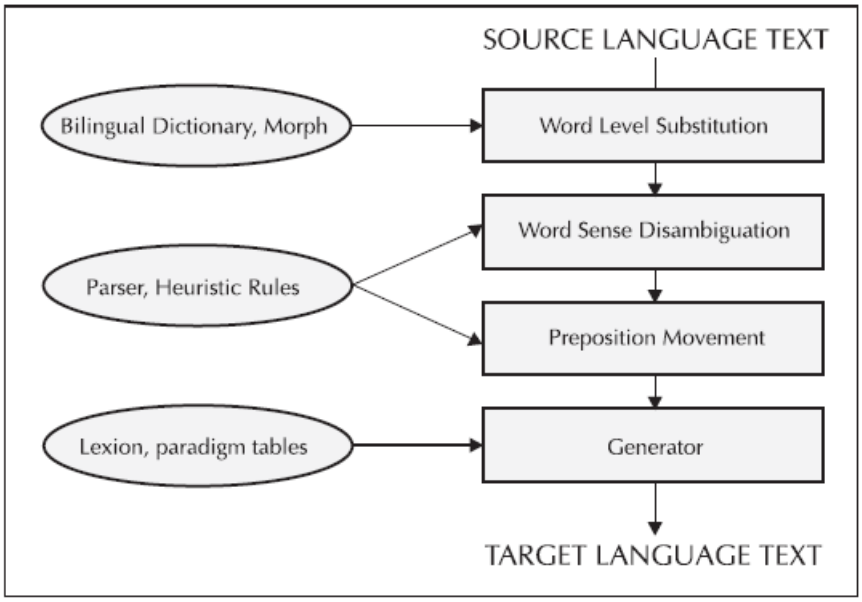


Figure 1 –The architecture of “core” anusaaraka

Architecture of the Anusaaraka system:

The Anusaaraka system has two major components.

_ Core engine
_ User–cum-developer interface

‘Core’ engine is the main engine of anusaaraka. This engine produces the output in different layers making the process of Machine Translation transparent to the user.

The architecture of “core” anusaaraka is shown in Figure 1.

This architecture differs from the conventional architecture in three major ways:

1. The order of operations is reversed. In the new architecture there is initial word level substitution followed by use of other language resources that are less reliable, like POS taggers, parsers, etc.

2. A graphical user interface has been developed to display the spectrum of outputs. The user has flexibility to adjust the output as per his/her needs. There will be users of different kinds based on the level of sophistication required and skill in handling the tool.

3. Special "interfaces", which act as 'glue' have been developed for different parsers, which allow plugging in of different parsers thereby providing modularity.

Core Anusaaraka engine

The core anusaaraka engine has four major modules viz.
I. Word Level Substitution
II. Word Sense Disambiguation
III. Preposition placement
IV. Word Order generation

I.Word Level Substitution

At this level the 'gloss' of each source language word into the target language is provided. However, the Polysemous words (words having more than one related meaning) create problems. When there is no one-one mapping, it is not practical to list all the meanings. On the other hand, anusaaraka claims 'faithfulness' to the original text. Then how is the faithfulness guaranteed at word level substitution?

II.Word Sense Disambiguation (WSD)

English has a very rich source of systematic ambiguity. Majority of nouns in English can potentially be used as verbs. Therefore, the WSD task in case of English can be split into two classes:
(i) WSD across POS
(ii) WSD within POS
The POS taggers can help in WSD when the ambiguity is across POSs. For example: Consider the two sentences 'He chairs the session'. 'The chairs in this room are comfortable'. The POS taggers mark the words with appropriate POS tags. These taggers use certain heuristic rules, and hence may sometimes go wrong. The reported performances of these POS taggers vary between 95% to 97%. However, they are still useful, since they reduce the search space for meanings substantially.

III.Preposition Placement

English has prepositions whereas Hindi has postpositions. Hence, it is necessary to move the prepositions to proper positions in Hindi before substituting their meanings. While moving the prepositions from their English positions to the proper Hindi positions, \record of their movements must be stored, so that in case a need arises, they can be reverted back to their original position. Therefore, the transformations performed by this module, are also reversible.

IV. Word Order Generation

Hindi is a free word order language. Therefore, even the anusaaraka output in the previous layer

makes sense to the Hindi reader. However, this output not being natural in Hindi, may not be enjoyed as much as the output with natural Hindi order. Additionally, it would not be treated as a translation. Therefore, in this module the attempt is to generate the correct Hindi word order.

Interface for different linguistic tools

The second major contribution of this architecture is the concept of 'interfaces'. Machine translation requires language resources such as POS taggers, morphological analyzers, and parsers. More than one kinds of each of these tools exist. Hence, it is wise to use these tools. However, there are problems.

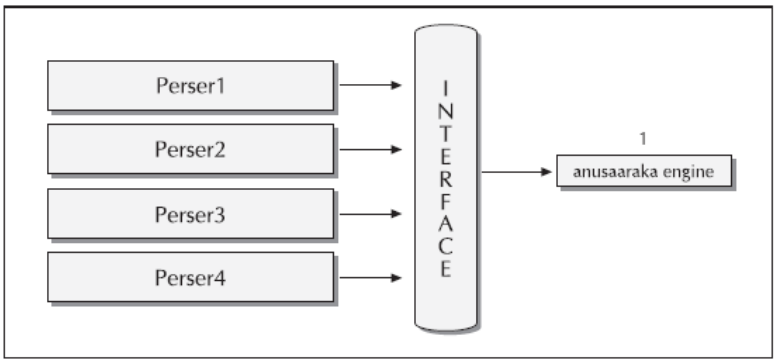


Figure 2 –Interfaces that map output of parsers to an intermediate form

As a machine translation system developer who is interested in the "usable" product one would like to plug-in different parsers and watch the performance. May be one would like to use combinations of them, or may like to vote among different parsers and choose the best parse out of them.
The Java/PYTHON based user interface has been developed to display the outputs produced by different layers of anusaaraka engine. The user interface provides a flexibility to control the display.

7.Multilingual Information Retrieval(MLIR)

Multilingual Information Retrieval (MLIR) is a subfield of Natural Language Processing (NLP) that focuses on retrieving relevant information from multilingual sources. It involves techniques and methodologies for searching, retrieving, and ranking documents or information in different languages.

The main goal of MLIR is to overcome language barriers and enable users to retrieve information from a diverse range of languages, even if they are not proficient in those languages. MLIR systems typically involve the following key components:

1. Multilingual Indexing: MLIR systems index documents from multiple languages to create a searchable collection. This process involves language-specific preprocessing techniques such as tokenization, stemming, and stop-word removal.
2. Cross-lingual Mapping: MLIR often involves creating mappings between different languages to establish connections and similarities. This can be achieved through techniques such as bilingual dictionaries, parallel corpora, or statistical models that learn cross-lingual word representations.
3. Query Translation: MLIR systems handle queries in one language and translate them into the languages of the indexed documents. Query translation methods include statistical machine translation, rule-based translation, or leveraging cross-lingual word embeddings.
4. Cross-lingual Retrieval Models: MLIR employs retrieval models that consider the multilingual nature of the indexed documents and queries. These models often combine language-specific relevance signals with cross-lingual information, such as document similarity or query expansion techniques.
5. Evaluation Metrics: MLIR systems are evaluated using metrics that account for the effectiveness of information retrieval across multiple languages. Common evaluation measures include mean average precision, precision at K, or cross-lingual variants of these metrics.

Challenges in MLIR include handling language-specific nuances, limited availability of resources for some languages, handling code-switching and mixed-language content, and scalability in indexing and retrieval for large multilingual collections.

MLIR finds applications in various domains such as cross-lingual search engines, multilingual digital libraries, e-commerce platforms, and information retrieval in multilingual social media content.

Researchers and practitioners in MLIR continue to explore advanced techniques, leveraging deep learning, neural networks, and transformer-based models to improve the effectiveness and efficiency of multilingual information retrieval systems.

8. Cross-Lingual Information Retrieval (CLIR)

CLIR stands for Cross-Language Information Retrieval, and it is a subfield of Natural Language Processing (NLP) that focuses on retrieving information across different languages. It involves the process of searching for and retrieving relevant documents or information in a target language, given a query expressed in a different source language.

The goal of CLIR is to bridge the language barrier and enable users to access information from different languages, even if they do not understand or speak those languages. It is particularly useful in multilingual and cross-cultural contexts, where people may need to search for information in languages they are not familiar with.

CLIR typically involves the following steps:

1. **Query Translation**: The user query, expressed in the source language, needs to be translated into the target language. This step can be challenging due to differences in grammar, vocabulary, and linguistic structure between languages.
2. **Document Indexing**: The documents in the target language need to be indexed to enable efficient retrieval. This typically involves extracting relevant features from the documents, such as keywords, named entities, or language-specific patterns.
3. **Retrieval**: The translated query is used to search the indexed documents, and retrieval algorithms rank the documents based on their relevance to the query. Various information retrieval techniques, such as vector space models or probabilistic models, can be applied here.
4. **Result Presentation**: The retrieved documents are presented to the user, often with additional processing to provide a summary, highlight relevant information, or support further exploration.

CLIR faces several challenges due to the inherent complexities of language translation, variations in language resources and structures, and the scarcity of parallel or bilingual data. Researchers in the field employ various techniques to address these challenges, including statistical machine translation, cross-lingual word embeddings, and leveraging multilingual resources such as dictionaries or parallel corpora.

CLIR has applications in areas such as multilingual search engines, digital libraries, cross-cultural communication, and global information access. It enables users to overcome language barriers and access information from diverse linguistic sources, fostering knowledge dissemination and collaboration across different language communities.

---

9. Evaluation in Information Retrieval

Evaluation in Information Retrieval (IR) in NLP refers to the process of assessing and measuring the effectiveness and performance of IR systems or models in retrieving relevant information in response to user queries. Evaluation plays a crucial role in assessing the quality of IR systems, comparing different approaches, and driving improvements in the field. There are several commonly used evaluation measures in IR:

1. **Precision:** Precision measures the proportion of retrieved documents that are relevant to a given query. It is calculated as the number of relevant documents retrieved divided by the total number of documents retrieved.
2. **Recall**: Recall measures the proportion of relevant documents that are retrieved out of all the relevant documents available in the collection. It is calculated as the number of relevant documents retrieved divided by the total number of relevant documents.
3. **F1 Score**: The F1 score combines precision and recall into a single metric, providing a balanced measure of system performance. It is the harmonic mean of precision and recall, calculated as 2 * (precision * recall) / (precision + recall).
4. **Mean Average Precision (MAP):** MAP is a widely used measure for ranked retrieval. It calculates the average precision across different recall levels for a set of queries. It considers the order in which documents are retrieved and rewards systems that retrieve relevant documents earlier in the ranked list.

5. **Normalized Discounted Cumulative Gain (NDCG):** NDCG is a measure that accounts for the relevance and rank of retrieved documents. It assigns higher scores to relevant documents that are ranked higher in the list. NDCG takes into account both precision and the position of relevant documents in the ranked list.
6. **Precision at K:** Precision at K measures the precision of the top-K retrieved documents. It considers only the first K documents and calculates the proportion of relevant documents among them.
7. **Mean Reciprocal Rank (MRR):** MRR measures the rank at which the first relevant document is retrieved. It calculates the reciprocal of the rank and takes the average across multiple queries.

These evaluation measures are used in experimental settings where a set of queries and relevant documents are predefined. The effectiveness of an IR system is evaluated by comparing its performance against a ground truth set of relevant documents. Additionally, evaluation may also involve user studies, where human assessors judge the relevance of retrieved documents based on their expertise or preferences.

It's important to note that evaluation in IR is an ongoing area of research, and different evaluation measures may be used depending on the specific task, dataset, or application. Researchers and practitioners continually work to develop new evaluation techniques that better reflect user needs and system performance in real-world scenarios.

10. Tools, Software and Resources

There are numerous tools, software libraries, and resources available for Natural Language Processing (NLP) that can assist with various NLP tasks. Here are some commonly used ones:

1. **NLTK (Natural Language Toolkit):** NLTK is a popular open-source library for NLP written in Python. It provides a wide range of functionalities and tools for tasks such as tokenization, stemming, tagging, parsing, and classification. It also offers access to corpora, lexical resources, and pre-trained models.
2. **spaCy**: spaCy is a Python library for advanced NLP tasks. It provides efficient tokenization, named entity recognition, part-of-speech tagging, dependency parsing, and lemmatization. spaCy focuses on performance and is known for its speed and ease of use.
3. **Gensim**: Gensim is a Python library for topic modeling, document similarity analysis, and other unsupervised NLP tasks. It offers implementations of popular algorithms like Latent Semantic Analysis (LSA), Latent Dirichlet Allocation (LDA), and Word2Vec.
4. **Stanford CoreNLP**: Stanford CoreNLP is a suite of NLP tools developed by Stanford University. It offers a wide range of capabilities, including tokenization, part-of-speech tagging, named entity recognition, dependency parsing, sentiment analysis, and coreference resolution. CoreNLP supports multiple languages and provides Java APIs along with wrappers for other programming languages.
5. **TensorFlow**: TensorFlow is an open-source deep learning framework that includes tools and libraries for NLP. It provides a high-level API called TensorFlow Hub, which offers pre-trained models for tasks like text classification, machine translation, and text generation. TensorFlow also supports the development of custom NLP models using deep learning architectures.
6. **PyTorch**: PyTorch is another popular deep learning framework that offers support for NLP. It provides tools for building and training neural networks, including modules for text classification, sequence labeling,

and language generation. PyTorch also offers pre-trained models, such as BERT and GPT, for various NLP tasks.

7. **WordNet**: WordNet is a lexical database that organizes words into sets of synonyms called synsets. It also provides semantic relationships between words, such as hypernyms, hyponyms, and meronyms. WordNet is widely used for tasks like word sense disambiguation, lexical similarity, and semantic analysis.
8. **Word Embeddings**: Word embeddings are distributed representations of words in a continuous vector space. Pre-trained word embeddings, such as Word2Vec, GloVe, and FastText, are available for download and can be used to capture semantic relationships between words in NLP models.
9. **Universal Dependencies**: Universal Dependencies is a project that provides syntactic annotation standards for a large number of languages. It offers pre-annotated treebanks that represent the syntactic structure of sentences, which can be used for tasks like parsing and dependency analysis.
10. **BERT** (Bidirectional Encoder Representations from Transformers): BERT is a pre-trained deep learning model that has achieved state-of-the-art performance on various NLP tasks, including question answering, named entity recognition, and sentiment analysis. The original BERT model and its variations are available for fine-tuning and transfer learning.

These are just a few examples of the many tools, software libraries, and resources available for NLP. The choice of tools depends on the specific task, programming language preference, and the complexity of the project at hand. It's important to explore and experiment with different tools to find the ones that best suit your needs.

11. Multilingual Automatic Summarization

Multilingual automatic summarization refers to the process of automatically generating concise summaries of text documents written in multiple languages. It involves extracting the most important information from a given text and presenting it in a condensed form, while maintaining the essential meaning and context across different languages.

There are several approaches to multilingual automatic summarization:

1. **Statistical Methods**: These methods analyze the statistical properties of the text, such as word frequency and distribution, to identify key sentences or phrases for summarization. Techniques like TF-IDF (Term Frequency-Inverse Document Frequency) and graph-based algorithms are commonly used.
2. **Machine Learning**: Machine learning techniques, including supervised and unsupervised algorithms, are employed to train models on multilingual datasets. These models learn to recognize important content and generate summaries based on patterns observed in the training data.
3. **Cross-Lingual Transfer Learning**: This approach involves leveraging pre-trained language models that have been trained on large multilingual corpora. These models can understand and

generate text in multiple languages, enabling them to perform summarization tasks across different languages without needing language-specific training data.

4. **Alignment-based Methods**: These methods align sentences or phrases in different languages to find corresponding content, allowing for the creation of summaries that preserve the meaning and coherence across languages. Techniques like parallel text alignment and cross-lingual sentence embeddings are used in this approach.
5. **Hybrid Approaches**: Some systems combine multiple techniques to achieve better performance in multilingual summarization tasks. For example, a system might use statistical methods for sentence extraction and machine learning models for content scoring and summarization generation.

Multilingual automatic summarization has various applications, including language translation, cross-lingual information retrieval, and facilitating access to multilingual content on the web. It helps users quickly grasp the main points of documents written in different languages, thereby improving efficiency and accessibility in multilingual communication and information processing.

11. Explain Approaches to Summarization in NLP

In Natural Language Processing (NLP), summarization refers to the process of condensing a text document while retaining its most important information. There are several approaches to automatic summarization in NLP, each with its own techniques and methodologies:

1. **Extractive Summarization**:
    - Extractive summarization involves selecting and combining important sentences or phrases directly from the original text to form a summary.
    - Techniques such as TextRank, a graph-based algorithm similar to Google's PageRank, and algorithms based on clustering and sentence scoring are commonly used.
    - Extractive methods are relatively simpler and preserve the wording of the original text but may lack coherence and fluency.
2. **Abstractive Summarization**:
    - Abstractive summarization involves generating new sentences that capture the essence of the original text in a more concise form.
    - This approach often utilizes deep learning techniques, such as Recurrent Neural Networks (RNNs), Transformer-based models (like BERT and GPT), and sequence-to-sequence models (such as LSTM or GRU with attention mechanisms).

- Abstractive methods can produce more coherent and fluent summaries compared to extractive methods but may struggle with maintaining factual accuracy and generating grammatically correct sentences.

3. **Query-Based Summarization**:
   - Query-based summarization focuses on generating summaries based on specific queries or questions provided by the user.
   - It typically involves techniques like Information Retrieval (IR) to identify relevant passages or sentences from the text that address the query, followed by summarization methods to generate a concise answer.
   - Query-based approaches are useful for tasks such as question answering and providing contextually relevant summaries for user queries.

4. **Domain-Specific Summarization**:
   - Domain-specific summarization techniques are tailored to particular domains or types of documents, such as scientific articles, legal documents, or news articles.
   - These methods often incorporate domain-specific knowledge and terminology to improve the quality and relevance of the summaries.
   - Domain-specific summarization may involve specialized pre-processing steps, feature engineering, or fine-tuning of models on domain-specific datasets.
5. **Multi-Document Summarization**:
   - Multi-document summarization aims to generate a summary from a collection of multiple documents on a given topic or event.
   - Techniques for multi-document summarization include clustering similar documents, identifying important information common across documents, and fusion of summaries from individual documents.
   - Multi-document summarization is particularly useful for tasks such as literature reviews, news aggregation, and summarizing discussions on social media.

These approaches to summarization in NLP cater to various needs and applications, ranging from extracting key information from large volumes of text to generating concise summaries tailored to specific queries or domains. Depending on the task requirements and the nature of the input text, different summarization techniques may be more suitable.

12. Explain Summarization Evaluation

Summarization evaluation is the process of assessing the quality and effectiveness of automatic summarization systems. Evaluating summarization systems is crucial for determining their performance, comparing different approaches, and guiding improvements in summarization algorithms. Several evaluation metrics and methodologies are commonly used in the field of automatic summarization:

1. **ROUGE (Recall-Oriented Understudy for Gisting Evaluation)**:
    - ROUGE is one of the most widely used metrics for evaluating summarization systems.
    - It calculates the overlap between the n-grams (sequences of n words) in the generated summary and the reference (gold standard) summaries.
    - ROUGE-N measures overlap at the unigram, bigram, trigram, etc., levels, providing a comprehensive assessment of content overlap.
    - ROUGE-L and ROUGE-W consider the longest common subsequences and weighted LCS, respectively, between the summary and the reference.
    - ROUGE is effective for assessing the content overlap between the generated summary and human-written reference summaries.
2. **BLEU (Bilingual Evaluation Understudy)**:
    - BLEU was originally designed for machine translation evaluation but has been adapted for summarization evaluation.
    - It computes the precision of n-grams in the generated summary compared to the reference summaries.
    - BLEU considers the presence of n-grams in both the generated and reference summaries, focusing on fluency and grammaticality.
    - However, BLEU may penalize summaries that differ syntactically or structurally from the reference summaries, which can be problematic for abstractive summarization systems.
3. **METEOR (Metric for Evaluation of Translation with Explicit Ordering)**:
    - METEOR evaluates summarization systems by considering the precision, recall, and alignment between the generated and reference summaries.
    - It incorporates stemming, synonymy, and paraphrasing to account for variations in word choice and sentence structure.
    - METEOR is particularly useful for assessing summaries that may differ in wording or phrasing from the reference summaries.
4. **Human Evaluation**:
    - Human evaluation involves having human judges assess the quality of summaries produced by the summarization system.
    - Judges may rate summaries based on criteria such as informativeness, coherence, readability, and relevance to the original text.
    - Human evaluation provides valuable insights into the overall quality and usability of summaries but can be resource-intensive and subjective.
5. **Task-Specific Evaluation**:
    - Task-specific evaluation measures assess the performance of summarization systems based on their effectiveness for specific downstream tasks.
    - For example, in question-answering tasks, the relevance and completeness of the generated summaries in answering questions can be evaluated.
    - Task-specific evaluation provides a practical assessment of the utility of summaries for specific applications.

Evaluating summarization systems often involves a combination of these metrics and methodologies to provide a comprehensive understanding of their performance. The choice of evaluation metrics depends on factors such as the nature of the summarization task, the availability of reference summaries, and the desired characteristics of the generated summaries.

13. Explain How to Build a Summarizer in NLP

Building a summarizer in Natural Language Processing (NLP) involves several steps, from preprocessing the text data to implementing the summarization algorithm. Here's a high-level overview of the process:

1. **Preprocessing**:
   - Clean the text data: Remove irrelevant characters, punctuation, and special symbols.
   - Tokenization: Split the text into individual words or tokens.
   - Sentence segmentation: Split the text into sentences to process them independently.
2. **Feature Extraction**:
   - Calculate sentence importance scores: Assign importance scores to each sentence based on criteria such as word frequency, position in the document, or semantic similarity to other sentences.
   - Compute word embeddings: Represent words in the text as dense vectors in a high-dimensional space to capture their semantic meanings.
3. **Summarization Algorithm**:
   - Extractive Summarization:
     - TextRank Algorithm: Apply the TextRank algorithm, a graph-based ranking algorithm similar to Google's PageRank, to identify important sentences based on their connectivity within the document.
     - Clustering: Cluster sentences based on similarity measures such as cosine similarity or Jaccard similarity and select representative sentences from each cluster as the summary.
     - Machine Learning Models: Train machine learning models (e.g., Support Vector Machines, Random Forests) on labeled data to predict sentence importance scores and select top-ranked sentences as the summary.
   - Abstractive Summarization:
     - Sequence-to-Sequence Models: Implement sequence-to-sequence models, such as Recurrent Neural Networks (RNNs) with attention mechanisms or Transformer-based architectures (e.g., BERT, GPT), to generate summaries by paraphrasing and rephrasing the input text.
     - Reinforcement Learning: Train models using reinforcement learning techniques to generate summaries by maximizing rewards based on predefined criteria (e.g., ROUGE scores, semantic similarity).
4. **Post-processing**:
   - Reconstruct the summary: Combine selected sentences or generated phrases to form a coherent summary.
   - Ensure summary length: Limit the length of the summary to meet specific requirements or constraints.
   - Remove redundant information: Filter out redundant sentences or phrases to improve the quality and conciseness of the summary.
5. **Evaluation**:
   - Evaluate the performance of the summarization model using appropriate evaluation metrics such as ROUGE, BLEU, METEOR, or human judgment.
   - Fine-tune the model parameters or adjust the summarization approach based on evaluation results to improve performance.

6. **Deployment and Integration**:
    - Deploy the summarization model as a standalone application or integrate it into existing NLP pipelines or platforms.
    - Provide an interface for users to input text documents and receive summarized outputs.

Throughout the process, it's essential to iterate on the design, experiment with different techniques, and continuously evaluate and refine the summarization model to achieve optimal performance and usability. Additionally, considering ethical implications such as fairness, bias, and privacy is crucial when building NLP applications like summarizers.

14. Explain Competitions and Datasets in NLP

Competitions and datasets play vital roles in advancing the field of Natural Language Processing (NLP) by providing benchmarks for evaluating and comparing algorithms, fostering innovation, and promoting collaboration among researchers and practitioners. Here's an overview of competitions and datasets in NLP:

1. **Competitions**:
    - **Shared Tasks**: Many NLP competitions are organized as shared tasks, where participants are given specific challenges or problems to solve using natural language processing techniques. These tasks can range from text classification and sentiment analysis to machine translation and summarization.
    - **Evaluation Metrics**: Competitions typically define evaluation metrics to assess the performance of participants' solutions objectively. Common metrics include accuracy, precision, recall, F1-score, and task-specific metrics like ROUGE for summarization or BLEU for machine translation.
    - **Platforms**: Competitions are often hosted on platforms like Kaggle, SemEval (Semantic Evaluation), and the Conference on Computational Natural Language Learning (CoNLL). These platforms provide infrastructure for organizing competitions, submitting solutions, and benchmarking results.
    - **Community Engagement**: Competitions encourage collaboration and knowledge sharing within the NLP community. Participants often publish their approaches and findings, leading to advancements in the field.
    - **Example Competitions**: Some notable NLP competitions include the SemEval tasks, the General Language Understanding Evaluation (GLUE) benchmark, the Conversational Intelligence Challenge (ConvAI), and the Text REtrieval Conference (TREC).
2. **Datasets**:
    - **Annotated Datasets**: Annotated datasets are essential for training and evaluating NLP models. They contain text documents or corpora annotated with labels, annotations, or ground truth information for specific tasks, such as sentiment analysis, named entity recognition, or question answering.
    - **Large-Scale Datasets**: Large-scale datasets, such as the Common Crawl, Wikipedia dumps, and the BooksCorpus, provide vast amounts of text data for training deep learning models. These datasets enable researchers to develop models with better generalization and scalability.
    - **Task-Specific Datasets**: Datasets are tailored to specific NLP tasks, such as sentiment analysis (e.g., the IMDb dataset), machine translation (e.g., the WMT datasets), and question answering (e.g., the SQuAD

dataset). Task-specific datasets facilitate benchmarking and comparison of models across different domains and applications.

- **Multilingual Datasets**: Multilingual datasets contain text data in multiple languages, allowing researchers to develop models that can understand and process text in different languages. Examples include the Multi30K dataset for image captioning and the XNLI dataset for cross-lingual natural language inference.
- **Ethical Considerations**: It's crucial to consider ethical considerations when creating and using datasets in NLP, such as ensuring privacy, avoiding bias and discrimination, and obtaining informed consent from data subjects.

Both competitions and datasets are instrumental in driving progress and innovation in NLP, enabling researchers and practitioners to develop more accurate, robust, and scalable natural language processing solutions.

**********END**********