Module -4

Natural Language Processing :

Developing programs to understand natural language is important in AI because a natural form of communication with systems is essential for user acceptance. One of the most critical tests for intelligent behavior is the ability to communicate effectively. This was the test proposed by Alan Turing. AI programs must be able to communicate with their human counterparts in a natural way, and natural language is one of the most important mediums for that purpose. A program understands a natural language if it behaves by taking a correct or acceptable action in response to the input. For example, we say a child demonstrates understanding if it responds with the correct answer to a question. The action taken need not be the external response. It may be the creation of some internal data structures. The structures created should be meaningful and correctly interact with the world model representation held by the program. In this chapter we explore many of the important issues related to natural language understanding and language generation.

This chapter explores several techniques that are used to enable humans to interact with computers via natural human languages. Natural languages are the languages used by humans for communication (among other functions). They are distinctly different from formal languages, such as C++, Java, and PROLOG. One of the main differences, which we will examine in some detail in this chapter, is that natural languages are ambiguous, meaning that a given sentence can have more than one possible meaning, and in some cases the correct meaning can be very hard to determine. Formal languages are almost always designed to ensure that ambiguity cannot occur. Hence, a given program written in C++ can have only one interpretation. This is clearly desirable because otherwise the computer would have to make an arbitrary decision as to which interpretation to work with. It is becoming increasingly important for computers to be able to understand natural languages. Telephone systems are now widespread that are able to understand a narrow range of commands and questions to assist callers to large call centers, without needing to use human resources. Additionally, the quantity of unstructured textual data that exists in the world (and in particular, on the Internet) has reached unmanageable proportions. For humans to search through these data using traditional techniques such as Boolean queries or the database query language SQL is impractical. The idea that people should be able to pose questions in their own language, or something similar to it, is an increasingly popular one. Of course, English is not the only natural language. A great deal of research in natural language processing and information retrieval is carried out in English, but many human languages differ enormously from English. Languages such as Chinese, Finnish, and Navajo have almost nothing in common with English (although of course Finnish uses the same alphabet). Hence, a system that can work with one human language cannot necessarily deal with any other human language. In this section we will explore two main topics. First, we will examine natural language processing, which is a collection of techniques used to enable computers to "understand" human language. In general, they are concerned with extracting grammatical information as well as meaning from human utterances but they are also concerned with understanding those utterances, and performing useful tasks as a result. Two of the earliest goals of natural language processing were automated translation (which is explored in this chapter) and database access. The idea here was that if a user wanted to find some information from a database, it would

make much more sense if he or she could query the database in her language, rather than needing to learn a new formal language such as SQL. Information retrieval is a collection of techniques used to try to match a query (or a command) to a set of documents from an existing corpus of documents. Systems such as the search engines that we use to find data on the Internet use information retrieval (albeit of a fairly simple nature).

Overview of linguistics

In dealing with natural language, a computer system needs to be able to process and manipulate language at a number of levels.

- Phonology. This is needed only if the computer is required to understand spoken language. Phonology is the study of the sounds that make up words and is used to identify words from sounds. We will explore this in a little more detail later, when we look at the ways in which computers can understand speech.
- Morphology. This is the first stage of analysis that is applied to words, once they have been identified from speech, or input into the system. Morphology looks at the ways in which words break down into components and how that affects their

grammatical status. For example, the letter “s” on the end of a word can often either indicate that it is a plural noun or a third-person present-tense verb.

- Syntax. This stage involves applying the rules of the grammar from the language being used. Syntax determines the role of each word in a sentence and, thus, enables a computer system to convert sentences into a structure that can be more easily manipulated.
- Semantics. This involves the examination of the meaning of words and sentences. As we will see, it is possible for a sentence to be syntactically correct but to be semantically meaningless. Conversely, it is desirable that a computer system be able to understand sentences with incorrect syntax but that still convey useful information semantically.
- Pragmatics. This is the application of human-like understanding to sentences and discourse to determine meanings that are not immediately clear from the semantics. For example, if someone says, “Can you tell me the time?”, most people know that “yes” is not a suitable answer. Pragmatics enables a computer system to give a sensible answer to questions like this.
- In addition to these levels of analysis, natural language processing systems must apply some kind of world knowledge. In most real-world systems, this world knowledge is limited to a specific domain (e.g., a system might have detailed knowledge about the Blocks World and be able to answer questions about this world). The ultimate goal of natural language processing would be to have a system with enough world knowledge to be able to engage a human in discussion on any subject. This goal is still a long way off.
- **Morphological Analysis**
    - In studying the English language, morphology is relatively simple. We have endings such as -ing, -s, and -ed, which are applied to verbs; endings such as -s and -es, which are applied to nouns; we also have the ending -ly, which usually indicates that a word is an adverb.
    - We also have prefixes such as anti-, non-, un-, and in-, which tend to indicate negation, or opposition.
    - We also have a number of other prefixes and suffixes that provide a variety of semantic and syntactic information.

- In practice, however, morphologic analysis for the English language is not terribly complex, particularly when compared with agglutinative languages such as German, which tend to combine words together into single words to indicate combinations of meaning.
- Morphologic analysis is mainly useful in natural language processing for identifying parts of speech (nouns, verbs, etc.) and for identifying which words belong together.
- In English, word order tends to provide more of this information than morphology, however. In languages such as Latin, word order was almost entirely superficial, and the morphology was extremely important. Languages such as French, Italian, and Spanish lie somewhere between these two extremes.
- As we will see in the following sections, being able to identify the part of speech for each word is essential to understanding a sentence. This can partly be achieved by simply looking up each word in a dictionary, which might contain for example the following entries:

  (swims, verb, present, singular, third person)

  (swimmer, noun, singular)

  (swim, verb, present, singular, first and second persons)

  (swim, verb, present plural, first, second, and third persons)

  (swimming, participle)

  (swimmingly, adverb)

  (swam, verb, past)

- Clearly, a complete dictionary of this kind would be unfeasibly large. A more practical approach is to include information about standard endings, such as:

  (-ly, adverb)

  (-ed, verb, past)

  (-s, noun, plural)

- This works fine for regular verbs, such as walk, but for all natural languages there are large numbers of irregular verbs, which do not follow these rules. Verbs such as to be and to do are particularly difficult in English as they do not seem to follow any morphologic rules.
- The most sensible approach to morphologic analysis is thus to include a set of rules that work for most regular words and then a list of irregular words.
- For a system that was designed to converse on any subject, this second list would be extremely long. Most natural language systems currently are designed to discuss fairly limited domains and so do not need to include over-large look-up tables.
- In most natural languages, as well as the problem posed by the fact that word order tends to have more importance than morphology, there is also the difficulty of ambiguity at a word level.
- This kind of ambiguity can be seen in particular in words such as trains, which could be a plural noun or a singular verb, and set, which can be a noun, verb, or adjective.

- **BNF**
  - Parsing involves mapping a linear piece of text onto a hierarchy that represents the way the various words interact with each other syntactically.
  - First, we will look at grammars, which are used to represent the rules that define how a specific language is built up.
  - Most natural languages are made up of a number of parts of speech, mainly the following:
    - Verb
    - Noun
    - Adjective
    - Adverb
    - Conjunction
    - Pronoun
    - Article
  - In fact it is useful when parsing to combine words together to form syntactic groups. Hence, the words, a dog, which consist of an article and a noun, can also be described as a noun phrase.

- A noun phrase is one or more words that combine together to represent an object or thing that can be described by a noun. Hence, the following are valid noun phrases: christmas, the dog, that packet of chips, the boy who had measles last year and nearly died, my favorite color
- A noun phrase is not a sentence—it is part of a sentence.
- A verb phrase is one or more words that represent an action. The following are valid verb phrases: swim, eat that packet of chips, walking
- A simple way to describe a sentence is to say that it consists of a noun phrase and a verb phrase. Hence, for example: That dog is eating my packet of chips.
- In this sentence, that dog is a noun phrase, and is eating my packet of chips is a verb phrase. Note that the verb phrase is in fact made up of a verb phrase, is eating, and a noun phrase, my packet of chips.
- A language is defined partly by its grammar. The rules of grammar for a language such as English can be written out in full, although it would be a complex process to do so.
- To allow a natural language processing system to parse sentences, it needs to have knowledge of the rules that describe how a valid sentence can be constructed.
- These rules are often written in what is known as Backus–Naur form (also known as Backus normal form—both names are abbreviated as BNF).
- BNF is widely used by computer scientists to define formal languages such as C++ and Java. We can also use it to define the grammar of a natural language.
- A grammar specified in BNF consists of the following components:
  - Terminal symbols. Each terminal symbol is a symbol or word that appears in the language itself. In English, for example, the terminal symbols are our dictionary words such as the, cat, dog, and so on. In formal languages, the terminal symbols include variable names such as x, y, and so on, but for our purposes we will consider the terminal symbols to be the words in the language.
  - Nonterminal symbols. These are the symbols such as noun, verb phrase, and conjunction that are used to define words and phrases of

the language. A nonterminal symbol is so-named because it is used to represent one or more terminal symbols.

- The start symbol. The start symbol is used to represent a complete sentence in the language. In our case, the start symbol is simply sentence, but in first-order predicate logic, for example, the start symbol would be expression.
- Rewrite rules. The rewrite rules define the structure of the grammar. Each rewrite rule details what symbols (terminal or nonterminal) can be used to make up each nonterminal symbol.

- Let us now look at rewrite rules in more detail. We saw above that a sentence could take the following form: noun phrase verb phrase
- We thus write the following rewrite rule: Sentence→NounPhrase VerbPhrase This does not mean that every sentence must be of this form, but simply that a string of symbols that takes on the form of the right-hand side can be rewritten in the form of the left-hand side. Hence, if we see the words

  The cat sat on the mat

- we might identify that the cat is a noun phrase and that sat on the mat is a verb phrase. We can thus conclude that this string forms a sentence.
- We can also use BNF to define a number of possible noun phrases.
- Note how we use the “|” symbol to separate the possible right-hand sides in BNF:

  NounPhrase→ Noun

  | Article Noun

  | Adjective Noun

  | Article Adjective Noun

- Similarly, we can define a verb phrase:

  VerbPhrase→ Verb

  | Verb NounPhrase

  | Adverb Verb NounPhrase

- The structure of human languages varies considerably. Hence, a set of rules like this will be valid for one language, but not necessarily for any other language.
- For example, in English it is usual to place the adjective before the noun (black cat, stale bread), whereas in French, it is often the case that the adjective comes after the noun (moulin rouge). Thus far, the rewrite rules we have written consist solely of nonterminal symbols.
- Rewrite rules are also used to describe the parts of speech of individual words (or terminal symbols):

```
Noun→ cat
| dog
| Mount Rushmore
| chickens
Verb→ swims
| eats
| climbs
Article→ the
| a
Adjective→ black
| brown
| green
| stale
```

Grammars and Languages

- The types of grammars that exist are Noam Chomsky invented a hierarchy of grammars.

- The hierarchy consists of four main types of grammars.
- The simplest grammars are used to define regular languages.
- A regular language is one that can be described or understood by a finite state automaton. Such languages are very simplistic and allow sentences such as "aaaaabbbbbb." Recall that a finite state automaton consists of a finite number of states, and rules that define how the automaton can transition from one state to another.
- A finite state automaton could be designed that defined the language that consisted of a string of one or more occurrences of the letter a. Hence, the following strings would be valid strings in this language:

  aaa

  a

  aaaaaaaaaaaaaaaaa

- Regular languages are of interest to computer scientists, but are not of great interest to the field of natural language processing because they are not powerful enough to represent even simple formal languages, let alone the more complex natural languages.
- Sentences defined by a regular grammar are often known as regular expressions.
- The grammar that we defined above using rewrite rules is a context-free grammar.
- It is context free because it defines the grammar simply in terms of which word types can go together—it does not specify the way that words should agree with each.

  A stale dog climbs Mount Rushmore.

- It also, allows the following sentence, which is not grammatically correct:

  Chickens eats.

- A context-free grammar can have only at most one terminal symbol on the right-hand side of its rewrite rules.
- Rewrite rules for a context-sensitive grammar, in contrast, can have more than one terminal symbol on the right-hand side. This enables the grammar to specify number, case, tense, and gender agreement.
- Each context-sensitive rewrite rule must have at least as many symbols on the right-hand side as it does on the left-hand side.

- Rewrite rules for context-sensitive grammars have the following form:

  A X B→A Y B

  which means that in the context of A and B, X can be rewritten as Y.

- Each of A, B, X, and Y can be either a terminal or a nonterminal symbol.
- Context-sensitive grammars are most usually used for natural language processing because they are powerful enough to define the kinds of grammars that natural languages use. Unfortunately, they tend to involve a much larger number of rules and are a much less natural way to describe language, making them harder for human developers to design than context free grammars.
- The final class of grammars in Chomsky's hierarchy consists of recursively enumerable grammars (also known as unrestricted grammars).
- A recursively enumerable grammar can define any language and has no restrictions on the structure of its rewrite rules. Such grammars are of interest to computer scientists but are not of great use in the study of natural language processing.
- Parsing: Syntactic Analysis
  - As we have seen, morphologic analysis can be used to determine to which part of speech each word in a sentence belongs. We will now examine how this information is used to determine the syntactic structure of a sentence.
  - This process, in which we convert a sentence into a tree that represents the sentence's syntactic structure, is known as parsing.
  - Parsing a sentence tells us whether it is a valid sentence, as defined by our grammar
  - If a sentence is not a valid sentence, then it cannot be parsed. Parsing a sentence involves producing a tree, such as that shown in Fig 10.1, which shows the parse tree for the following sentence:

    The black cat crossed the road.

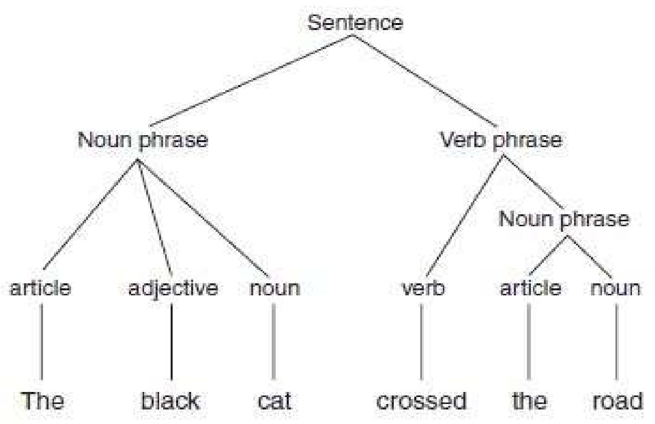


Fig 10.1

- This tree shows how the sentence is made up of a noun phrase and a verb phrase.
- The noun phrase consists of an article, an adjective, and a noun. The verb phrase consists of a verb and a further noun phrase, which in turn consists of an article and a noun.
- Parse trees can be built in a bottom-up fashion or in a top-down fashion.
- Building a parse tree from the top down involves starting from a sentence and determining which of the possible rewrites for Sentence can be applied to the sentence that is being parsed. Hence, in this case, Sentence would be rewritten using the following rule:

  Sentence→NounPhrase VerbPhrase

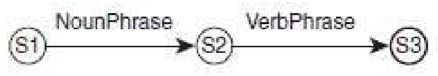
- Then the verb phrase and noun phrase would be broken down recursively in the same way, until only terminal symbols were left.
- When a parse tree is built from the top down, it is known as a derivation tree.
- To build a parse tree from the bottom up, the terminal symbols of the sentence are first replaced by their corresponding nonterminals (e.g., cat is replaced by noun), and then these nonterminals are combined to match the right-hand sides of rewrite rules.
- For example, the and road would be combined using the following rewrite rule: NounPhrase→Article Noun

Basic parsing techniques

- Transition Networks
    - A transition network is a finite state automaton that is used to represent a part of a grammar.
    - A transition network parser uses a number of these transition networks to represent its entire grammar.
    - Each network represents one nonterminal symbol in the grammar. Hence, in the grammar for the English language, we would have one transition network for Sentence, one for Noun Phrase, one for Verb Phrase, one for Verb, and so on.
    - Fig 10.2 shows the transition network equivalents for three production rules.
    - In each transition network, S1 is the start state, and the accepting state, or final state, is denoted by a heavy border. When a phrase is applied to a transition network, the first word is compared against one of the arcs leading from the first state.
    - If this word matches one of those arcs, the network moves into the state to which that arc points. Hence, the first network shown in Fig 10.2, when presented with a Noun Phrase, will move from state S1 to state S2.
    - If a phrase is presented to a transition network and no match is found from the current state, then that network cannot be used and another network must be tried. Hence, when starting with the phrase the cat sat on the mat, none of the networks shown in Fig 10.2 will be used because they all have only nonterminal symbols, whereas all the symbols in the cat sat on the mat are terminal.Hence, we need further networks, such as the ones shown in Figure 10.2, which deal with terminal symbols.

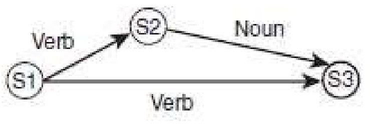
Production Rule | **Transition Network**

Sentence → NounPhrase VerbPhrase



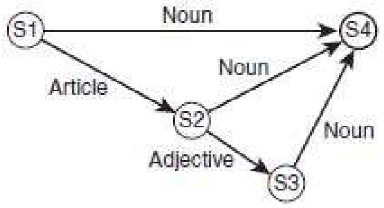
Sentence

VerbPhrase → Verb
| Verb Noun



VerbPhrase

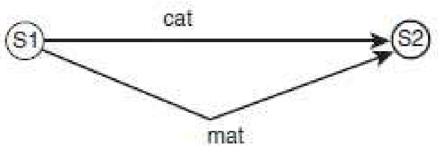
NounPhrase → Noun
| Article Noun
| Article Adjective Noun



NounPhrase

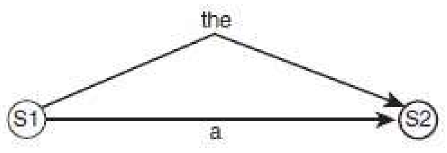
Production Rule | **Transition Network**

Noun → cat
| mat



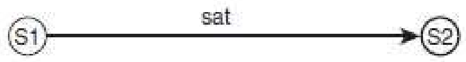
Noun

Article → the
| a



Article

Verb → sat



Verb

Fig 10.2

- Transition networks can be used to determine whether a sentence is grammatically correct, at least according to the rules of the grammar the networks represent.
- Parsing using transition networks involves exploring a search space of possible parses in a depth-first fashion.
- Let us examine the parse of the following simple sentence:

  A cat sat.

- We begin in state S1 in the Sentence transition network. To proceed, we must follow the arc that is labeled NounPhrase. We thus move out of the Sentence network and into the NounPhrase network.
- The first arc of the NounPhrase network is labeled Noun. We thus move into the Noun network. We now follow each of the arcs in the Noun network and discover that our first word, A, does not match any of them. Hence, we backtrack to the next arc in the NounPhrase network.
- This arc is labeled Article, so we move on to the Article transition network. Here, on examining the second label, we find that the first word is matched by the terminal symbol on this arc.
- We therefore consume the word, A, and move on to state S2 in the Article network. Because this is a success node, we are able to return to the NounPhrase network and move on to state S2 in this network. We now have an arc labeled Noun.
- As before, we move into the Noun network and find that our next word, cat, matches. We thus move to state S4 in the NounPhrase network. This is a success node, and so we move back to the Sentence network and repeat the process for the VerbPhrase arc.
- It is possible for a system to use transition networks to generate a derivation tree for a sentence, so that as well as determining whether the sentence is grammatically valid, it parses it fully to obtain further information by semantic analysis from the sentence.
- This can be done by simply having the system build up the tree by noting which arcs it successfully followed. When, for example, it successfully follows the NounPhrase arc in the Sentence network, the system generates a root node labeled Sentence and an arc leading from that node to a new node

labeled NounPhrase.When the system follows the NounPhrase network and identifies an article and a noun, these are similarly added to the tree.

- In this way, the full parse tree for the sentence can be generated using transition networks. Parsing using transition networks is simple to understand, but is not necessarily as efficient or as effective as we might hope for. In particular, it does not pay any attention to potential ambiguities or the need for words to agree with each other in case, gender, or number.

- **Augmented Transition Networks**
    - An augmented transition network, or ATN, is an extended version of a transition network.
    - ATNs have the ability to apply tests to arcs, for example, to ensure agreement with number. Thus, an ATN for Sentence would be as shown in Figure 10.2, but the arc from node S2 to S3 would be conditional on the number of the verb being the same as the number for the noun.
    - Hence, if the noun phrase were three dogs and the verb phrase were is blue, the ATN would not be able to follow the arc from node S2 to S3 because the number of the noun phrase (plural) does not match the number of the verb phrase (singular).
    - In languages such as French, checks for gender would also be necessary. The conditions on the arcs are calculated by procedures that are attached to the arcs. The procedure attached to an arc is called when the network reaches that arc. These procedures, as well as carrying out checks on agreement, are able to form a parse tree from the sentence that is being analyzed.

- **Chart Parsing**
    - Parsing using transition networks is effective, but not the most efficient way to parse natural language. One problem can be seen in examining the following two sentences: 1. Have all the fish been fed? , Have all the fish.
    - Clearly these are very different sentences—the first is a question, and the second is an instruction. In spite of this, the first threewords of each sentence are the same.

- When a parser is examining one of these sentences, it is quite likely to have to backtrack to the beginning if it makes the wrong choice in the first case for the structure of the sentence. In longer sentences, this can be a much greater problem, particularly as it involves examining the same words more than once, without using the fact that the words have already been analyzed.

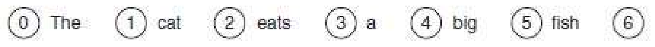


Fig 10.3

- Another method that is sometimes used for parsing natural language is chart parsing.
- In the worst case, chart parsing will parse a sentence of n words in O(n3) time. In many cases it will perform better than this and will parse most sentences in O(n2) or even O(n) time.
- In examining sentence 1 above, the chart parser would note that the words two children form a noun phrase. It would note this on its first pass through the sentence and would store this information in a chart, meaning it would not need to examine those words again on a subsequent pass, after backtracking.
- The initial chart for the sentence The cat eats a big fish is shown in Fig 10.3 It shows the chart that the chart parse algorithm would start with for parsing the sentence.
- The chart consists of seven vertices, which will become connected to each other by edges. The edges will show how the constituents of the sentence combine together.
- The chart parser starts by adding the following edge to the chart:

  [0, 0, Target→• Sentence]

- This notation means that the edge connects vertex 0 to itself (the first two numbers in the square brackets show which vertices the edge connects).
- Target is the target that we want to find, which is really just a placeholder to enable us to have an edge that requires us to find a whole sentence. The arrow indicates that in order to make what is on its left-hand side (Target) we need to find what is on its right-hand side (Sentence). The dot (•) shows

what has been found already, on its left-hand side, and what is yet to be found, on its right-hand side. This is perhaps best explained by examining an example.

- Consider the following edge, which is shown in the chart in Figure 10.4:

  [0, 2, Sentence→NounPhrase • VerbPhrase]

- This means that an edge exists connecting nodes 0 and 2. The dot shows us that we have already found a NounPhrase (the cat) and that we are looking for a VerbPhrase.

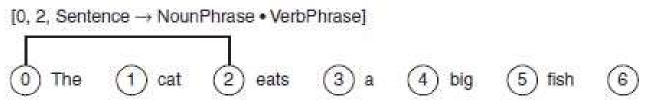

[0, 2, Sentence → NounPhrase • VerbPhrase]
0 The 1 cat 2 eats 3 a 4 big 5 fish 6

Fig 10.4

- Once we have found the VerbPhrase, we will have what is on the left-hand side of the arrow—that is, a Sentence.
- The chart parser can add edges to the chart using the following three rules:
  - If we have an edge [x, y, A →□B • C], which needs to find a C, then an edge can be added that supplies that C (i.e., the edge [x, y, C→ • E]), where E is some sequence of terminals or nonterminals which can be replaced by a C).
  - If we have two edges, [x, y, A →□B • C D] and [y, z, C →□E •}, then these two edges can be combined together to form a new edge: [x, z, A→B C • D].
  - If we have an edge [x, y, A →□B • C], and the word at vertex y is of type C, then we have found a suitable word for this edge, and so we extend the edge along to the next vertex by adding the following edge: [y, y + 1, A→B C •].

- **Semantic Analysis**
  - Having determined the syntactic structure of a sentence, the next task of natural language processing is to determine the meaning of the sentence.
  - Semantics is the study of the meaning of words, and semantic analysis is the analysis we use to extract meaning from utterances.

- Semantic analysis involves building up a representation of the objects and actions that a sentence is describing, including details provided by adjectives, adverbs, and prepositions. Hence, after analyzing the sentence The black cat sat on the mat, the system would use a semantic net such as the one shown in Figure 10.5 to represent the objects and the relationships between them.

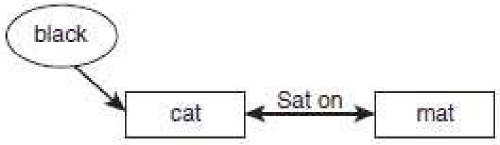


Fig 10.5

- A more sophisticated semantic network is likely to be formed, which includes information about the nature of a cat (a cat is an object, an animal, a quadruped, etc.) that can be used to deduce facts about the cat (e.g., that it likes to drink milk).

- Ambiguity and Pragmatic Analysis
  - One of the main differences between natural languages and formal languages like C++ is that a sentence in a natural language can have more than one meaning. This is ambiguity—the fact that a sentence can be interpreted in different ways depending on who is speaking, the context in which it is spoken, and a number of other factors.
  - The more common forms of ambiguity and look at ways in which a natural language processing system can make sensible decisions about how to disambiguate them.
  - Lexical ambiguity occurs when a word has more than one possible meaning. For example, a bat can be a flying mammal or a piece of sporting equipment. The word set is an interesting example of this because it can be used as a verb, a noun, an adjective, or an adverb. Determining which part of speech is intended can often be achieved by a parser in cases where only one analysis is possible, but in other cases semantic disambiguation is needed to determine which meaning is intended.
  - Syntactic ambiguity occurs when there is more than one possible parse of a sentence. The sentence Jane carried the girl with the spade could be

interpreted in two different ways, as is shown in the two parse trees in Fig 10.6. In the first of the two parse trees in Fig 10.6, the prepositional phrase with the spade is applied to the noun phrase the girl, indicating that it was the girl who had a spade that Jane carried. In the second sentence, the prepositional phrase has been attached to the verb phrase carried the girl, indicating that Jane somehow used the spade to carry the girl.

- Semantic ambiguity occurs when a sentence has more than one possible meaning—often as a result of a syntactic ambiguity. In the example shown in Fig 10.6 for example, the sentence Jane carried the girl with the spade, the sentence has two different parses, which correspond to two possible meanings for the sentence. The significance of this becomes clearer for practical systems if we imagine a robot that receives vocal instructions from a human.

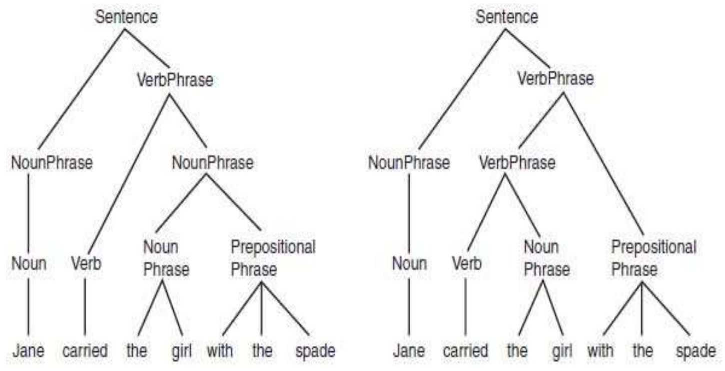


Fig 10.6

- Referential ambiguity occurs when we use anaphoric expressions, or pronouns to refer to objects that have already been discussed. An anaphora occurs when a word or phrase is used to refer to something without naming it. The problem of ambiguity occurs where it is not immediately clear which object is being referred to. For example, consider the following sentences:

  John gave Bob the sandwich. He smiled.

- It is not at all clear from this who smiled—it could have been John or Bob. In general, English speakers or writers avoid constructions such as this to avoid humans becoming confused by the ambiguity. In spite of this, ambiguity can also occur in a similar way where a human would not have a problem, such as

John gave the dog the sandwich. It wagged its tail.

- In this case, a human listener would know very well that it was the dog that wagged its tail, and not the sandwich. Without specific world knowledge, the natural language processing system might not find it so obvious.
- A local ambiguity occurs when a part of a sentence is ambiguous; however, when the whole sentence is examined, the ambiguity is resolved. For example, in the sentence There are longer rivers than the Thames, the phrase longer rivers is ambiguous until we read the rest of the sentence, than the Thames.
- Another cause of ambiguity in human language is vagueness. we examined fuzzy logic, words such as tall, high, and fast are vague and do not have precise numeric meanings.
- The process by which a natural language processing system determines which meaning is intended by an ambiguous utterance is known as disambiguation.
- Disambiguation can be done in a number of ways. One of the most effective ways to overcome many forms of ambiguity is to use probability.
- This can be done using prior probabilities or conditional probabilities. Prior probability might be used to tell the system that the word bat nearly always means a piece of sporting equipment.
- Conditional probability would tell it that when the word bat is used by a sports fan, this is likely to be the case, but that when it is spoken by a naturalist it is more likely to be a winged mammal.
- Context is also an extremely important tool in disambiguation. Consider the following sentences:

  I went into the cave. It was full of bats.

  I looked in the locker. It was full of bats.

- In each case, the second sentence is the same, but the context provided by the first sentence helps us to choose the correct meaning of the word "bat" in each case.
- Disambiguation thus requires a good world model, which contains knowledge about the world that can be used to determine the most likely

meaning of a given word or sentence. The world model would help the system to understand that the sentence Jane carried the girl with the spade is unlikely to mean that Jane used the spade to carry the girl because spades are usually used to carry smaller things than girls. The challenge, of course, is to encode this knowledge in a way that can be used effectively and efficiently by the system.

- The world model needs to be as broad as the sentences the system is likely to hear. For example, a natural language processing system devoted to answering sports questions might not need to know how to disambiguate the sporting bat from the winged mammal, but a system designed to answer any type of question would.

Expert System Architecture

An expert system is a set of programs that manipulate encoded knowledge to solve problems in a specialized domain that normally requires human expertise. An expert system's knowledge is obtained from expert sources and coded in a form suitable for the system to use in its inference or reasoning processes. The expert knowledge must be obtained from specialists or other sources of expertise, such as texts, journal, articles and databases. This type of knowledge usually requires much training and experience in some specialized field such as medicine, geology, system configuration, or engineering design. Once a sufficient body of expert knowledge has been auquired, it must be encoded in some form, loaded into a knowledge base, then tested, and refined continually throughout the life of the system

Characteristics Features of Expert Systems

Expert systems differ from conventional computer system in several important ways

1. Expert systems use knowledge rather than data to control the solution process. Much of the knowledge used in heuristic in nature rather than algorithmic
2. The knowledge is encoded and maintained as an entity separate from the aontrol program. As such, it is not complicated together with the control program itself. This permits the incremental addition and modification of the knowledge base without recompilation of the control programs. Furthermore, it is possible in some cases to use different knowledge bases with the same control programs to produce different types

of expert systems. Such systems are known as expert system shells since they may be loaded with different knowledge bases

3. Expert systems are capable of explaining how a particular conclusion was reached, and why requested information is needed during a consultation. This is important as it gives the user a chance to assess and understand the systems reasoning ability, thereby improving the user's confidence in the system
4. Expert systems use symbolic representations for knowledge and perform their inference through symbolic computations that closely resemble manipulations of natural language
5. Expert systems often reason with metaknowledge, that is, they reason with knowledge about themselves, and their own knowledge limits and capabilities

Rules for Knowledge Representation

- One way to represent knowledge is by using rules that express what must happen or what does happen when certain conditions are met.
- Rules are usually expressed in the form of IF . . . THEN . . . statements, such as: IF A THEN B This can be considered to have a similar logical meaning as the following: $A \rightarrow B$
- A is called the antecedent and B is the consequent in this statement.
- In expressing rules, the consequent usually takes the form of an action or a conclusion.
- In other words, the purpose of a rule is usually to tell a system (such as an expert system) what to do in certain circumstances, or what conclusions to draw from a set of inputs about the current situation.
- In general, a rule can have more than one antecedent, usually combined either by AND or by OR (logically the same as the operators $\wedge$ and $\vee$).
- Similarly, a rule may have more than one consequent, which usually suggests that there are multiple actions to be taken.
- In general, the antecedent of a rule compares an object with a possible value, using an operator.
- For example, suitable antecedents in a rule might be

    IF $x > 3$

IF name is “Bob”

IF weather is cold

- Here, the objects being considered are x, name, and weather; the operators are “>” and “is”, and the values are 3, “Bob,” and cold.
- Note that an object is not necessarily an object in the real-world sense—the weather is not a real world object, but rather a state or condition of the world.
- An object in this sense is simply a variable that represents some physical object or state in the real world.
- An example of a rule might be

IF name is “Bob”

AND weather is cold

THEN tell Bob ‘Wear a coat’

- This is an example of a recommendation rule, which takes a set of inputsand gives advice as a result.
- The conclusion of the rule is actually an action, and the action takes the form of a recommendation to Bob that he should wear a coat.
- In some cases, the rules provide more definite actions such as “move left” or “close door,” in which case the rules are being used to represent directives.
- Rules can also be used to represent relations such as:

IF temperature is below 0

THEN weather is cold

Rule-Based Systems

- **Rule-based systems** or **production systems** are computer systems that use rules to provide recommendations or diagnoses, or to determine a course of action in a particular situation or to solve a particular problem.
- A rule-based system consists of a number of components:
  - a database of rules (also called a **knowledge base**)
  - a database of facts

- an **interpreter**, or **inference engine**

- In a rule-based system, the knowledge base consists of a set of rules that represent the knowledge that the system has.
- The database of facts represents inputs to the system that are used to derive conclusions, or to cause actions.
- The interpreter, or inference engine, is the part of the system that controls the process of deriving conclusions. It uses the rules and facts, and combines them together to draw conclusions.
- Using deduction to reach a conclusion from a set of antecedents is called **forward chaining**.
- An alternative method, **backward chaining**, starts from a conclusion and tries to show it by following a logical path backward from the conclusion to a set of antecedents that are in the database of facts.
- **Forward Chaining**
    - Forward chaining employs the system starts from a set of facts, and a set of rules, and tries to find a way of using those rules and facts to deduce a conclusion or come up with a suitable course of action.
    - This is known as **data-driven reasoning** because the reasoning starts from a set of data and ends up at the goal, which is the conclusion.
    - When applying forward chaining, the first step is to take the facts in the fact database and see if any combination of these matches all the antecedents of one of the rules in the rule database.
    - When all the antecedents of a rule are matched by facts in the database, then this rule is **triggered**.
    - Usually, when a rule is triggered, it is then **fired**, which means its conclusion is added to the facts database. If the conclusion of the rule that has fired is an action or a recommendation, then the system may cause that action to take place or the recommendation to be made.
    - For example, consider the following set of rules that is used to control an elevator in a three-story building:

Rule 1

IF on first floor and button is pressed on first floor

THEN open door

Rule 2

IF on first floor

AND button is pressed on second floor

THEN go to second floor

Rule 3

IF on first floor

AND button is pressed on third floor

THEN go to third floor

Rule 4

IF on second floor

AND button is pressed on first floor

AND already going to third floor

THEN remember to go to first floor later

- This represents just a subset of the rules that would be needed, but we can use it to illustrate how forward chaining works.
- Let us imagine that we start with the following facts in our database:

Fact 1

At first floor

Fact 2

Button pressed on third floor

Fact 3

Today is Tuesday

- Now the system examines the rules and finds that Facts 1 and 2 match the antecedents of Rule 3. Hence, Rule 3 fires, and its conclusion “Go to third floor” is added to the database of facts. Presumably, this results in the elevator heading toward the third floor.
- Note that Fact 3 was ignored altogether because it did not match the antecedents of any of the rules.
- Now let us imagine that the elevator is on its way to the third floor and has reached the second floor, when the button is pressed on the first floor. The fact Button pressed on first floor
- Is now added to the database, which results in Rule 4 firing.
- Now let us imagine that later in the day the facts database contains the following information:

  Fact 1

  At first floor

  Fact 2

  Button pressed on second floor

  Fact 3

  Button pressed on third floor

- In this case, two rules are triggered—Rules 2 and 3. In such cases where there is more than one possible conclusion, **conflict resolution** needs to be applied to decide which rule to fire.

- **Conflict Resolution**
  - In a situation where more than one conclusion can be deduced from a set of facts, there are a number of possible ways to decide which rule to fire.
  - For example, consider the following set of rules:

    IF it is cold

    THEN wear a coat

    IF it is cold

THEN stay at home

IF it is cold

THEN turn on the heat

- If there is a single fact in the fact database, which is "it is cold," then clearly there are three conclusions that can be derived. In some cases, it might be fine to follow all three conclusions, but in many cases the conclusions are incompatible.
- In one conflict resolution method, rules are given priority levels, and when a conflict occurs, the rule that has the highest priority is fired, as in the following example:

IF patient has pain

THEN prescribe painkillers priority 10

IF patient has chest pain

THEN treat for heart disease priority 100

- Here, it is clear that treating possible heart problems is more important than just curing the pain.
- An alternative method is the **longest-matching strategy**. This method involves firing the conclusion that was derived from the longest rule.
- For example:

IF patient has pain

THEN prescribe painkiller

IF patient has chest pain

AND patient is over 60

AND patient has history of heart conditions

THEN take to emergency room

- Here, if all the antecedents of the second rule match, then this rule's conclusion should be fired rather than the conclusion of the first rule because it is a more specific match.
- A further method for conflict resolution is to fire the rule that has matched the facts most recently added to the database.
- In each case, it may be that the system fires one rule and then stops, but in many cases, the system simply needs to choose a suitable ordering for the rules because each rule that matches the facts needs to be fired at some point.

- **Meta Rules**
  - In designing an expert system, it is necessary to select the conflict resolution method that will be used, and quite possibly it will be necessary to use different methods to resolve different types of conflicts.
  - For example, in some situations it may make most sense to use the method that involves firing the most recently added rules.
  - This method makes most sense in situations in which the timeliness of data is important. It might be, for example, that as research in a particular field of medicine develops, and new rules are added to the system that contradicts some of the older rules.
  - It might make most sense for the system to assume that these newer rules are more accurate than the older rules.
  - It might also be the case, however, that the new rules have been added by an expert whose opinion is less trusted than that of the expert who added the earlier rules.
  - In this case, it clearly makes more sense to allow the earlier rules priority.
  - This kind of knowledge is called **meta knowledge**—knowledge about knowledge. The rules that define how conflict resolution will be used, and how other aspects of the system itself will run, are called **meta rules**.
  - The knowledge engineer who builds the expert system is responsible for building appropriate meta knowledge into the system (such as "expert A is to be trusted more than expert B" or "any rule that involves drug X is not to be trusted as much as rules that do not involve X").
  - Meta rules are treated by the expert system as if they were ordinary rules but are given greater priority than the normal rules that make up the expert system.

In this way, the meta rules are able to override the normal rules, if necessary, and are certainly able to control the conflict resolution process.

- **Backward Chaining**
    - Forward chaining applies a set of rules and facts to deduce whatever conclusions can be derived, which is useful when a set of facts are present, but you do not know what conclusions you are trying to prove.
    - Forward chaining can be inefficient because it may end up proving a number of conclusions that are not currently interesting.
    - In such cases, where a single specific conclusion is to be proved, **backward chaining** is more appropriate.
    - In backward chaining, we start from a conclusion, which is the **hypothesis** we wish to prove, and we aim to show how that conclusion can be reached from the rules and facts in the database.
    - The conclusion we are aiming to prove is called a **goal**, and so reasoning in this way is known as **goal-driven reasoning**.
    - Backward chaining is often used in formulating plans.
    - A plan is a sequence of actions that a program decides to take to solve a particular problem.
    - Backward chaining can make the process of formulating a plan more efficient than forward chaining.
    - Backward chaining in this way starts with the goal state, which is the set of conditions the agent wishes to achieve in carrying out its plan. It now examines this state and sees what actions could lead to it.
    - For example, if the goal state involves a block being on a table, then one possible action would be to place that block on the table.
    - This action might not be possible from the start state, and so further actions need to be added before this action in order to reach it from the start state.
    - In this way, a plan can be formulated starting from the goal and working back toward the start state.
    - The benefit in this method is particularly clear in situations where the first state allows a very large number of possible actions.
    - In this kind of situation, it can be very inefficient to attempt to formulate a plan using forward chaining because it involves examining every possible

action, without paying any attention to which action might be the best one to lead to the goal state.

- Backward chaining ensures that each action that is taken is one that will definitely lead to the goal, and in many cases this will make the planning process far more efficient.

- **Comparing Forward and Backward Chaining**
  - Let us use an example to compare forward and backward chaining. In this case, we will revert to our use of symbols for logical statements, in order to clarify the explanation, but we could equally well be using rules about elevators or the weather.

Rules:

Rule 1 $A \wedge B \rightarrow C$

Rule 2 $A \rightarrow D$

Rule 3 $C \wedge D \rightarrow E$

Rule 4 $B \wedge E \wedge F \rightarrow G$

Rule 5 $A \wedge E \rightarrow H$

Rule 6 $D \wedge E \wedge H \rightarrow I$

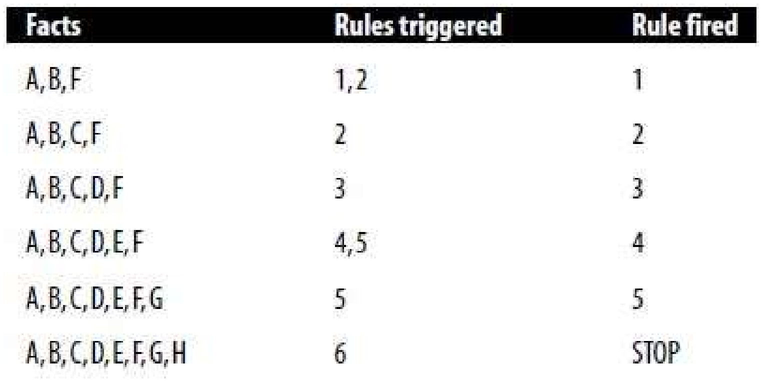
Facts:

Fact 1	A
Fact 2	B
Fact 3	F

Goal:

Our goal is to prove H.

- First let us use forward chaining. As our conflict resolution strategy, we will fire rules in the order they appear in the database, starting from Rule 1.
- In the initial state, Rules 1 and 2 are both triggered. We will start by firing Rule 1, which means we add C to our fact database. Next, Rule 2 is fired, meaning we add D to our fact database.
- We now have the facts A, B, C, D, F, but we have not yet reached our goal, which is G.
- Now Rule 3 is triggered and fired, meaning that fact E is added to the database.
- As a result, Rules 4 and 5 are triggered. Rule 4 is fired first, resulting in Fact G being added to the database, and then Rule 5 is fired, and Fact H is added to the database.
- We have now proved our goal and do not need to go on any further.
- This deduction is presented in the following table:

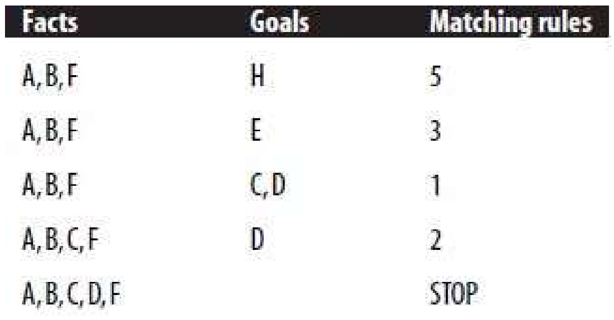


Facts	Rules triggered	Rule fired
A,B,F	1,2	1
A,B,C,F	2	2
A,B,C,D,F	3	3
A,B,C,D,E,F	4,5	4
A,B,C,D,E,F,G	5	5
A,B,C,D,E,F,G,H	6	STOP

- Now we will consider the same problem using backward chaining. To do so, we will use a goals database in addition to the rule and fact databases.
- In this case, the goals database starts with just the conclusion, H, which we want to prove. We will now see which rules would need to fire to lead to this conclusion.
- Rule 5 is the only one that has H as a conclusion, so to prove H, we must prove the antecedents of Rule 5, which are A and E.
- Fact A is already in the database, so we only need to prove the other antecedent, E. Therefore, E is added to the goal database. Once we have

proved E, we now know that this is sufficient to prove H, so we can remove H from the goals database.

- So now we attempt to prove Fact E. Rule 3 has E as its conclusion, so to prove E, we must prove the antecedents of Rule 3, which are C and D.
- Neither of these facts is in the fact database, so we need to prove both of them. They are both therefore added to the goals database. D is the conclusion of Rule 2 and Rule 2's antecedent, A, is already in the fact database, so we can conclude D and add it to the fact database.
- Similarly, C is the conclusion of Rule 1, and Rule 1's antecedents, A and B, are both in the fact database. So, we have now proved all the goals in the goal database and have therefore proved H and can stop.
- This process is represented in the table below:



Facts	Goals	Matching rules
A, B, F	H	5
A, B, F	E	3
A, B, F	C, D	1
A, B, C, F	D	2
A, B, C, D, F		STOP

- In this case, backward chaining needed to use one fewer rule. If the rule database had had a large number of other rules that had A, B, and F as their antecedents, then forward chaining might well have been even more inefficient.
- In general, backward chaining is appropriate in cases where there are few possible conclusions (or even just one) and many possible facts, not very many of which are necessarily relevant to the conclusion.
- Forward chaining is more appropriate when there are many possible conclusions.
- The way in which forward or backward chaining is usually chosen is to consider which way an expert would solve the problem. This is particularly appropriate because rule-based reasoning is often used in **expert systems**.

Rule-Based Expert Systems