

16

Natural Language Processing

16.1 Introduction

Natural Language Processing (NLP) is an important field of AI since both understanding and generation of natural languages require a lot of intelligence. *Understanding* basically refers to the process of mapping from a given input form (text or speech) into more immediately useful form. It is represented as a pair (source language, target representation) along with a mapping between elements of each to other. Target representation varies from one application to other. For example, in the case of translation, target representation refers to target language; on the other hand, in case of paragraph comprehension, target representation may refer to some form of semantic representation using which one can answer various queries regarding the source text. Basically, NLP involves processing of written text using computer models at lexical, syntactic, and semantic level. It also includes processing of spoken language that uses all the processing techniques required for written text plus knowledge about phonology with added information to handle ambiguities that arise in speech.

In this chapter, we will concentrate on written language processing techniques. It includes syntactic structures and various grammars with parsing methodologies, semantic models, definite clause grammar, case grammar and its application in machine translation. At the end of the chapter, we have briefly discussed the universal networking language (UNL) which is a language-independent generic model used for storing target expression.

16.2 Sentence Analysis Phases

A sentence is usually analyzed through different phases such as morphological (Lexical) analysis, syntactic analysis, semantic analysis, discourse, and pragmatic analysis. Each of these phases is briefly explained below:

Morphological Analysis: Morphological analysis process (MAP) is carried out initially on the natural language sentences; this method tries to extract root word from declined or inflectional form of word after removing suffices and prefixes. For example, getting the root 'push' from declined forms such as pushed, pushing, pushes, etc. In addition to this, it also assigns appropriate syntactic categories such as noun, verb, adjective, etc., to all words in the sentence.

Syntactic Analysis: This method of analysis uses the result of MAP to build a structural description of the sentence based on grammatical rules. This process is called *parsing*. A declarative representation (called *grammar*) of syntactic facts about the language and procedure (called *parser*) compares the grammar against the input sentence to produce parse structures. Creating a parse tree is the first step towards understanding a sentence.

Semantic Analysis: It creates a semantic structure by ascribing the literal meaning to a sentence using parse structure obtained in syntactic phase. It maps individual words into corresponding objects in the knowledge base and combines the words with each other using semantic rules. Our aim here is to produce meaning in some suitable representation scheme by using any of KR methods described in earlier chapter. Main purpose of semantic processing is the creation of target representation of the meaning of sentence.

If a natural language is used as an interface to another system (programs such as database), then target representation must be so chosen that it is understood by the system. So the design of target representation is driven by the program using it.

Pragmatics Analysis: It refers to intended meaning of a sentence used in different contexts. The context affects the interpretation of the sentence. For example, in the sentence "John saw Mike in the garden with a cat", there are two interpretations. First interpretation is that *John* was having the garden with a cat, there are two interpretations. First interpretation is that *John* was having the garden with a cat and saw *Mike* in the garden. Other interpretation might be that *John* saw *Mike* (with cat) in the garden. Seeing the context one can resolve to unique interpretation. If we have knowledge about *John* that he keeps pets then first interpretation will be more suitable.

Discourse Analysis: It refers to conversation between two or more individuals and the interpretation is based on the belief set at the time of conversation. Here the interpretation of spoken sentence will be based on the belief set of the people involved in conversation.

A large number of computational models for syntactic and semantic analysis have been developed but very little work has been done for developing pragmatic and discourse analysis models since various complexities are involved in understanding them. Linguists have done researches and developed theories but these are not feasible straightway for computational purposes. These two

areas are open for research in NLP. We will discuss various formalisms available for syntactic and semantic analysis.

16.3 Grammars and Parsers

In general, the term *parsing* refers to the process of analyzing an input sequence in order to determine its structure with respect to a given grammar. In the context of natural language processing, parsing implies analyzing a sentence syntactically to assign syntactic tags (subject, verb, object, and so on) to provide constituent structure (noun phrase, verb phrase etc.) or to characterize the syntactic relations between two words (i.e., dependency representations). Parsing is a fundamental requirement for many natural language processing applications such as machine translation, information extraction, natural language interfaces, speech recognition, etc. Parsing techniques are broadly divided into *rule-based parsing* and *statistical parsing*. The most commonly used technique of the two is rule-based parsing where the syntactic structure of language is provided in the form of linguistic rules which can be coded as production rules that are similar to context-free rules. Production rules are defined using non-terminal (symbols to be further expanded) and terminal symbols (direct symbols found in the language). Statistical parsing methods require large corpora and linguistic knowledge is represented as statistical parameters or probabilities, which may be used to parse a given sentence (Jurafsky D. & Martin J.H., 2000). In this chapter, we will concentrate on rule-based parsers. Once the grammar rules are defined, a sentence is parsed using the grammar and a tree kind of structure is built, if the sentence is syntactically correct. This tree is called a *parse tree*. Parsing can be done using two methods: top-down parsing and bottom-up parsing. A parser can use any of these parsing methods. Each one has its merits and demerits (Allen J., 1994).

Bottom-up parsing: In bottom-up parsing, we start with the words in the sentence and apply grammar rules in the backward direction until a single tree is produced whose root matches with the start symbol.

Top-down parsing: In top-down parsing, we start with the start symbol and apply grammar rules in the forward direction until the terminal symbols of the parse tree correspond to the words in the sentence.

Consider simple context-free-like grammar for English language as given in Table 16.1 The conventions used in this chapter are as follows:

- The symbol $\rightarrow$ is used for 'defined as'
- Vertical bar | for alternative definitions
- The symbols S for Sentence,
- NP for Noun phrase, and
- VP for Verb Phrase.

The declarative meaning of the rule $\langle S \rangle \rightarrow \langle NP \rangle \langle VP \rangle$ is that the sentence S can take a form in which *noun_phrase* (NP) is followed by a *verb_phrase* (VP). The simple sentences recognized by

the grammar given in Table 16.1 are *the boy eats an apple*, *the cute girl sings a song*, etc. Here, non-terminal symbols such as S, NP, VP, etc. are represented in capital letters, whereas terminal symbols (direct category words) such as Noun, Verb, Adj, etc. start with an uppercase letter followed by lowercase letters.

Table 16.1 Context-Free Grammar for a Subset of English Language

Rules		Dictionary Words	
<S>	→	<NP> <VP>	<Det> → a the an
<NP>	→	<Det> <Noun>	<Noun> → girl apple song
<NP>	→	<Det> <Adj> <Noun>	<Adj> → cute smart
<NP>	→	<Adj> <Noun>	<Verb> → sings eats ate
<VP>	→	<Verb>	
<VP>	→	<Verb> <NP>	

Let us apply both the methods for parsing English sentences:

Bottom-Up Parsing

Let us consider the following sentence: *The cute girl ate an apple* and see its parse structure (Fig. 16.1). The direction of the arrow shows the way in which the parse structure is made. Here, the syntactic category to each word in the sentence is assigned before starting the actual parsing. The application of rules is quite clear from the parse structure and needs no explanation.

Look-ahead Parser is basically a bottom-up parser, where the parser when deciding how to interpret input word is allowed to look at the next k input items before making its decision for an appropriate rule.

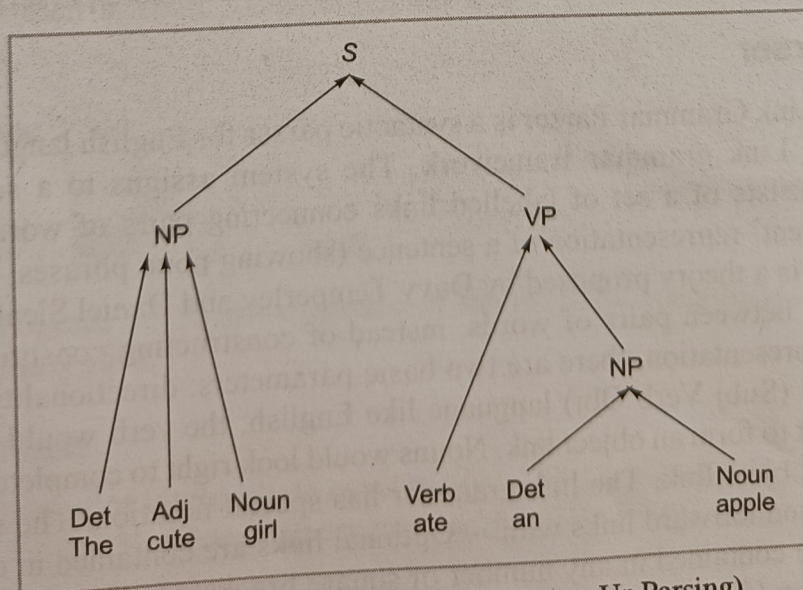


Figure 16.1 Parse Structure (Bottom-Up Parsing)

Top-Down Parsing

To parse a sentence using top-down approach, we start from the start symbol S and apply grammar rules in the forward direction until the terminal symbols of the parse tree corresponds to the words in the sentence (Fig. 16.2). This process may require backtracking for trying alternative rules if matching with the words of the sentence fail.

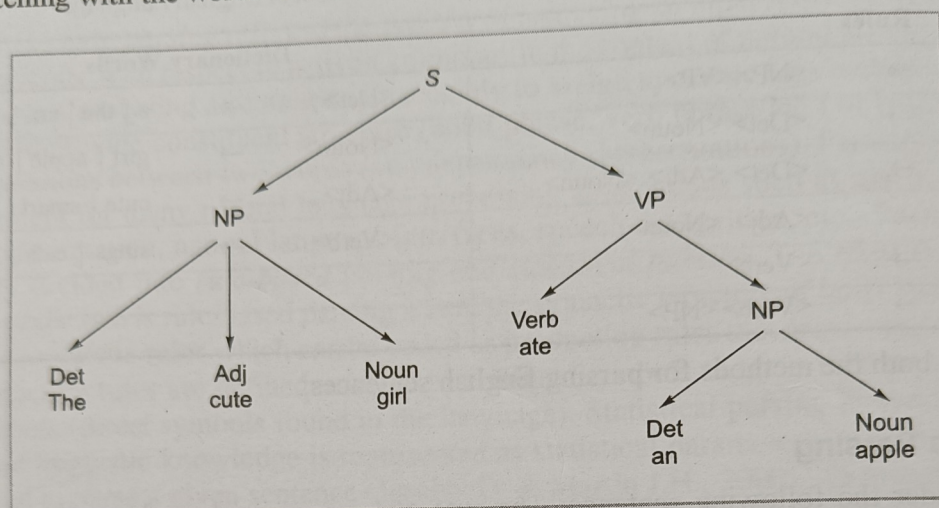


Figure 16.2 Parse Structure (Top-Down Parsing)

16.4 Types of Parsers

There are different types of parsers used for parsing sentences, namely, Link parser, Chart Parsers, Transition network parsers, Definite clause grammar, etc. Each of these parsers is described below.

16.4.1 Link Parser

The Link Parser or Link Grammar Parser is a syntactic parser for English language which analyzes sentences using the Link grammar framework. The system assigns to a sentence a syntactic structure, which consists of a set of labelled links connecting pairs of words. The parser also produces a 'constituent' representation of a sentence (showing noun phrases, verb phrases, etc.). *Link grammar* (LG) is a theory proposed by Davy Temperley and Daniel Sleator in the year 1993 that builds relations between pairs of words, instead of constructing constituents in a tree kind hierarchy. In this representation, there are two basic parameters: directionality and distance. For example, in an SVO (Subj Verb Obj) language like English, the verb would look left to form a subject link, and right to form an object link. Nouns would look right to complete the subject link, or left to complete the object link. The link grammar has special notation. The rightward links are represented with +, and leftward links with -. Optional links are contained in curly brackets {...}. Undesirable links are contained in any number of square brackets [...]. Multiple links are joined either by a conjunction (&) or a disjunction (or). Each rule ends with a semicolon. Just to give an idea about how link grammar looks, we are giving simple rules for SVO language.

- Basic rules for an SVO type language are given in the Table 16.2 with simple labels as *Sub* for subject, *Det* for determiner, *Obj* for object.

Table 16.2 Link Grammar Rules for SVO type Languages

SVO language	
Link Grammar Rules	Interpretation of Rules
<Determiner>: Det+;	Determiner is defined as label <i>Det</i> connected to word to its right in a sentence
<Noun_Sub>: {Det-} & Sub+;	Subject noun is defined as label <i>Sub</i> connected to word on its right in a sentence and connected to label <i>Det</i> on left in a sentence (it may be optional)
<Noun_Obj>: {Det-} & Obj-;	Object noun is defined as the last word in a sentence and labeled as <i>Obj</i> is connected to label <i>Det</i> on left in a sentence (it may be optional)
<Verb>: Sub- & {Obj+};	Verb is defined as a word connected to label <i>Obj</i> on right in a sentence (it may be optional) and connected to label <i>Sub</i> on left.

The parsing of English sentence, *The girl sings a song* using grammar given in Table 16.2 would appear as follows. Each token is assigned syntactic category of the word such as 'd' for determiner; 'n' for noun; 'v' for verb. Linkage diagram given below conveys the information about the sentence structure. Links between words are represented by the horizontal dashed lines. These are labelled with a link type. The parser also shows a constituent representation, derived from the linkage, of the sentence, labelling noun phrases, verb phrases, sentence ("S"), etc. The constituents are indented to show their level of embedding in the tree*.

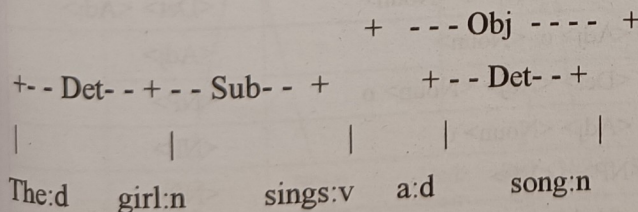


Chart Parsers: (Winograd, 1983) stores parses of intermediate constituents to be reused while trying alternative parsing path.

16.4.2 Chart Parser

Chart parsers store parses of intermediate constituents to be reused while trying alternative parsing path (Winograd T., 1983). Chart Parsers store intermediate constituents of parses that can be

*[For details refer to the link <http://www.link.cs.cmu.edu/link/>].

reused at the time of backtracking in bottom up parsing approach (Allen J.F., 1994). A data structure called *chart* is maintained to keep a record of the state of a bottom-up parse traversed. This structure is a record of the positions of the words and new structure derived from the sentence. Chart also maintains the record of rules that have been applied previously but are not complete. These rules are recorded as active arcs on the chart. Parsing using chart parsing method is shown below: Let us consider the CFG (Table 16.3) to be used for parsing the sentences.

Table 16.3 Grammar for a Subset of English language

Rules	Rule Number	Dictionary Words
<S> → <NP> <VP>	1	<Det> → a the an
<NP> → <Det> <Noun>	2	<Noun> → girl apple song
<NP> → <Det> <Adj> <Noun>	3	<Adj> → cute smart
<NP> → <Adj> <Noun>	4	<Verb> → sings ate
<VP> → <Verb>	5	
<VP> → <Verb> <NP>	6	

The chart parsing of the sentence *The cute girl sings a song* is shown in Table 16.4. The words in the sentence are numbered with corresponding positions such as

₁The ₂ cute ₃ girl ₄ sings ₅ a ₆ song ₇.

Table 16.4 Chart Parsing

Token	Active arc	Rule Number	Status of Rule Processing 'o' indicates position till parse is done	Completed Constituents
		1	<S> → o <NP> <VP>	
the	1 to 2	2	<NP> → <Det> o <Noun>	<Det>
	1 to 2	3	<NP> → <Det> o <Adj> <Noun>	<Det>
cute	1 to 3	3	<NP> → <Det> <Adj> o <Noun>	<Det> <Adj>
	2 to 3	4	<NP> → <Adj> o <Noun>	<Adj>
girl	1 to 4	3	<NP> → <Det> <Adj> <Noun> o	<NP>
	2 to 4	4	<NP> → <Adj> <Noun> o	<NP>
	1 to 4	1	<S> → <NP> o <VP>	<NP>
sings	4 to 5	5	<VP> → <Verb> o	<Verb>
	4 to 5	6	<VP> → <Verb> o <NP>	<Verb>
a	5 to 6	2	<NP> → <Det> o <Noun>	<Det>
	5 to 6	3	<NP> → <Det> o <Adj> <Noun>	<Det>
song	6 to 7	2	<NP> → <Det> <Noun> o	<NP>
	4 to 7	6	<VP> → <Verb> <NP> o	<VP>
	1 to 7	1	<S> → <NP> <VP> o	<S>

Table 16.4 contains tokens to be parsed, the active arcs are between words position, rule numbers applied with possible position of token parsed shown by character 'o'. For example, the interpretation of the rule ' $\langle NP \rangle \rightarrow \langle Det \rangle o \langle Noun \rangle$ ' is that 'Det' is found and looking for noun token. Similarly in the rule ' $\langle S \rangle \rightarrow \langle NP \rangle \langle VP \rangle o$ ' the sentence is found and no token further required.

Let us see the use of chart parsing where if we get more than one valid parse structures, then the constituents already computed are available for use in chart to generate new parse structure. There will be two valid parse structures for the sentence *the girls saw a man in the park with a cat* using the grammar defined below with prepositional phrases. Both parse structures are shown in Figures 16.3(a) and (b) using the following grammar using the grammar specified in Table 16.5.

Table 16.5 Sample Grammar

Grammar Rules	Rule Number
$\langle S \rangle \rightarrow \langle NP \rangle \langle VP \rangle$	1
$\langle NP \rangle \rightarrow \langle Det \rangle \langle Noun \rangle$	2
$\langle NP \rangle \rightarrow \langle Det \rangle \langle Noun \rangle \langle PP \rangle$	3
$\langle VP \rangle \rightarrow \langle Verb \rangle \langle NP \rangle$	4
$\langle VP \rangle \rightarrow \langle Verb \rangle \langle NP \rangle \langle PP \rangle$	5
$\langle PP \rangle \rightarrow \langle Prep \rangle \langle NP \rangle$	6

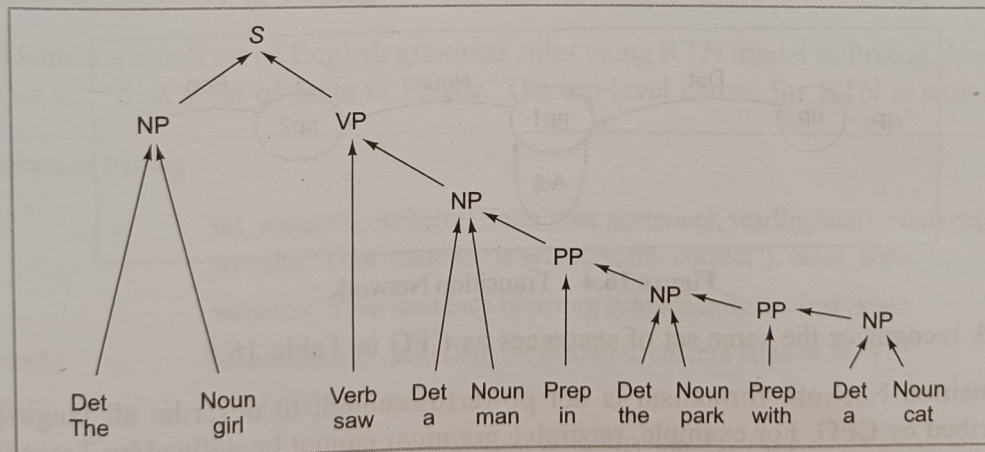


Figure 16.3(a) Parse Structure - I

The parse structures for the constituent NP(the, girls), NP(a, man, in, the, park), PP(with, NP(a, cat)) generated using chart parser in parse structure I can be easily utilized while creating parse structure II [Refer Fig. 16.3(b)].

There are other parsing processes that use transition network for transition from start state to final state corresponding to the grammar of the language. It includes Simple Transition Network, Recursive Transition Network (RTN), and Augmented Transition Network (ATN) (Woods, 1970).

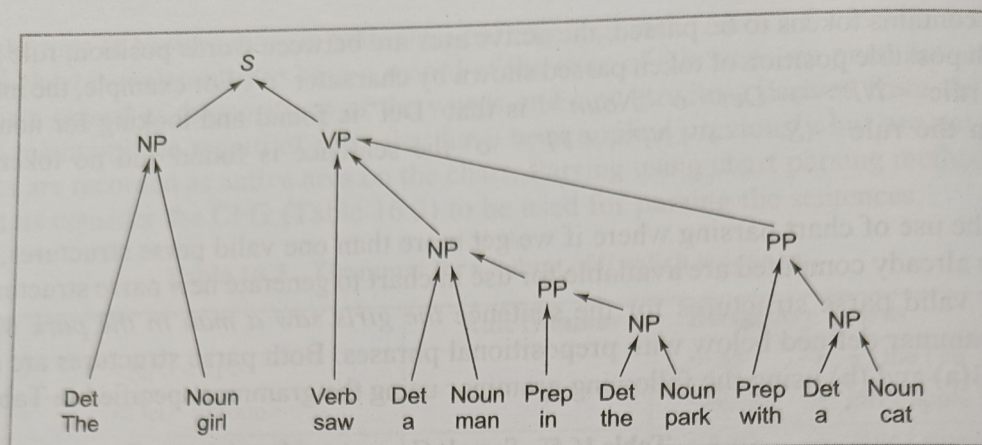


Figure 16.3(b) Parse Structure – II

16.4.3 Simple Transition Networks

Transition Network (TN) is more convenient for visualizing the grammar. It consists of nodes and labeled arcs. Parsing starts with the start node and traversal takes place through arcs. If the current word in the sentence is in the category on that arc then move to the next state in TN and continue the process till *Pop* arc is reached. If there are no words left in the sentence at *Pop* arc then it is assumed to be correctly parsed. Let us draw transition network for NP (Noun phrase). It consists of three nodes and arcs labeled by word categories *Det*, *Noun*, *Adj*. Figure 16.4 shows TN for NP.

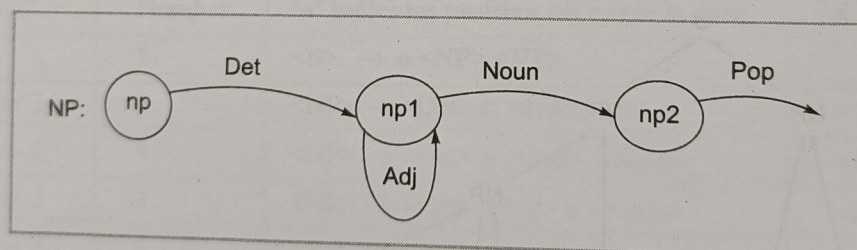


Figure 16.4 Transition Network

This network recognizes the same set of sentences as CFG in Table 16.6.

Simple Transition Network formalism is not powerful enough to describe all languages that can be described by CFG. For example, recursive grammar cannot be defined by Transition Net. For this purpose, TN is extended to Recursive Transition Network (RTN) as explained in the following subsection.

Table 16.6 Context-Free Grammar Corresponding to TN

$\langle \text{NP} \rangle$	$\rightarrow$	$\langle \text{Det} \rangle \langle \text{NP1} \rangle$
$\langle \text{NP1} \rangle$	$\rightarrow$	$\langle \text{Adj} \rangle \langle \text{NP1} \rangle$
$\langle \text{NP1} \rangle$	$\rightarrow$	$\langle \text{Noun} \rangle$

The lexicons are stored in a separate file (Table 16.20) and can be consulted while executing program containing grammar rules.

Table 16.20 Lexicons

Lexicons File
is_noun (singular, girl).
is_noun (plural, girls).
is_noun (singular, song).
is_noun(singular, banana).
is_noun(plural, apples).
is_noun(singular, orange).
is_verb(singular, eats).
is_verb(plural, eat).
is_verb(singular, sings).
is_verb(plural, sing).
is_det(singular, a).
is_det(singular, an).
is_det(_ , the).

16.5 Semantic Analysis

Semantic analysis basically involves analyzing sentence for its meaning. Generally, a parse structure created by syntactic parsing is analyzed further and a semantic structure is created. But in some approaches, syntactic and semantic phases are combined into one, the sentence if syntactically correct, will be transformed to its semantic structure directly. The main purpose of semantic analysis is the creation of target representation of the meaning of the sentence in a suitable representation. Therefore, a semantic structure is created by mapping individual words into corresponding objects in the knowledge base and combining the words with each other using semantic rules. The design of target language (semantic representation) is driven by the system using it. For example, if a language is used as an interface to another system (such as database, etc.), then target representation must be chosen that is understood by the system.

The first step in semantic processing is to look up the meaning of an individual word from the lexicon (dictionary). This phase is called *Lexical Processing*. It should be noted that the meaning of a word need not be unique. There might be more than one meaning for a word in the lexicon. The process of determining the correct meaning is called sense disambiguation or lexical disambiguation. This is done by associating with each word in the lexicon its context in which a particular sense is used. The context for each word in the sentence is the entire sentence or may be some text containing that sentence. For example, the word 'bank' can be used as an organization offering financial services or the land alongside of river. Therefore, the sense of word 'bank' is resolved by seeing the context. To solve lexical disambiguation problem, more and more

fine-grained *semantic markers* are to be used. Simple properties of word are called semantic markers. There are several approaches for creating semantic representation of a sentence. Some of these are:

- Semantic Grammars
- Case Grammar
- Conceptual parser
- Compositional semantic interpretation

16.5.1 Semantic Grammars

Combining syntactic and semantic processing involves collapsing them into a single uniform framework. This results in semantic grammar that is similar to context-free grammar incorporating semantic information in the syntactic rules. Semantic rules look like regular grammar except that it uses semantic categories for terminal symbols. Consider the following simple syntactic grammar represented as context-free-like grammar. Here, the category symbol enclosed in () indicates optional occurrence of that category (Table 16.21).

Table 16.21 Context-Free Grammar

<S>	→	<NP> <VP>
<NP>	→	(<Det>) <Noun> <Name>
<VP>	→	<Verb> <NP> (<PP>)
<PP>	→	<Prep> <NP>
<Det>	→	a the an
<Noun>	→	bus train man ticket
<Name>	→	John Mike Delhi Mumbai
<Verb>	→	books reserves leaves departs
<Prep>	→	to for from

Using this grammar, we can parse both syntactically correct and incorrect sentences as shown in Table 16.22.

Table 16.22 List of Syntactic Correct and Incorrect Sentences

Syntactically Correct sentences	Syntactically incorrect sentences
John reserves a ticket	The Train reserves to Delhi
Train leaves to Delhi	Train books from Mumbai
Train leaves from Delhi	A ticket reserves John
John reserves train to Delhi	Bus reserves John to Delhi
Man books ticket for bus to Delhi	Delhi books ticket for bus to Mike
Train departs Delhi	Delhi departs train

Table 16.23 Semantic Grammar (Extension of CFG)

<S>	→	<NP_A> <VP_Res>
<S>	→	<NP_Vehicle> <VP_Dep>
<NP_A>	→	(<Det>) <Noun_A> <P_Name>
<NP_Vehicle>	→	(<Det>) <Noun_Vehicle> (<PP>)
<VP_Res>	→	<Verb_Res> <NP_Vehicle>
<VP_Res>	→	<Verb_Res> <NP_NonA> for <NP_Vehicle>
<VP_Dep>	→	<Verb_Dep> <PP>
<PP>	→	to from <Loc>
<Det>	→	a the an
<Noun_A>	→	man
<P_Name>	→	John Mike
<Noun_NonA>	→	ticket
<Noun_Vehicle>	→	bus train flight
<Loc>	→	Delhi Mumbai
<Verb_Res>	→	books reserves
<Verb_Dep>	→	leaves departs

If we modify the grammar given in Table 16.21 using semantic information, then syntactically incorrect sentences could not be parsed. We clearly see that semantic grammar (Table 16.23) tends to be larger in size as compared to CFG and hence the efficiency of the parse degrades.

Alternatively, we can associate semantic action with syntactic rules. The parse tree for a sentence generated using such grammars generates the meaning of the sentence. Here, syntactic and semantic phases are combined and straight away we get semantic representation. Ambiguities in syntactic parse can be avoided by these grammars. One of the examples of such grammars is DCG grammar where we can incorporate semantic information in the rules and lexicons.

16.5.2 Semantic Grammar using DCG formalism

Using DCG formalism, we can get semantics in the form of first-order predicate logic. For this, we have to include one more argument in 'sentence' predicate which captures the semantic structure called logical form (LF) of a sentence. We will omit other arguments such as number and parse structure arguments discussed earlier (Subsection 16.46) for the sake of simplicity and will include only those arguments which are related to logical forms. Let us consider few examples and their corresponding logical forms.

- John is a man - man(john)
- Mary is a girl - girl(mary)
- Crow is a bird - bird(crow)

Let us write simple DCG grammar (Table 16.24) for handling sentences of the format 'noun is a noun' for returning LF while parsing.

Table 16.24 DCG for Generating LF (Sentence Format: Noun is a Noun)

DCG Rules			Rule Number
sentence(LF)	-->	np(N) , vp(V), {LF = V(N)}.	1
np(N)	-->	noun(N).	2
np(N)	-->	det, noun(N).	3
vp(V)	-->	aux_verb, np(V).	4
det	-->	[a].	5
aux_verb	-->	[is].	6
noun(man)	-->	[man].	7
noun(girl)	-->	[girl].	8
noun(crow)	-->	[crow].	9
noun(bird)	-->	[bird].	10
noun(john)	-->	[john].	11
noun(mary)	-->	[mary].	12
noun(mary)	-->	[mary].	13

The goal `?- sentence(LF, [john, is, man], [])` produces the result as `LF = man(john)`.

Let us write DCG grammar for handling the following complex sentences such as

- *every human is mortal* - $(\forall X) (human(X) \rightarrow mortal(X))$
- *every woman loves a child* - $(\forall X) (woman(X) \rightarrow (\$Y) (child(Y) \wedge likes(X, Y)))$

The Logical form for a sentence ' $(\forall X)(human(X) \rightarrow mortal(X))$ ' can be written as a term in prolog as ' $all(X, (human(X) \rightarrow mortal(X)))$ '.

Table 16.25 DCG for Generating LF (Sentence Format: Every Noun is Noun)

DCG Rules			Rule Number
sentence(LF)	-->	np(X, F, LF), vp(X, F).	1
np(X, F)	-->	noun(X, F).	2
np(X, F, LF)	-->	det(X, B, F, LF), noun(X, B).	3
vp(X, F)	-->	aux_verb, np(X, F).	4
det(X, B, F, all(X, (B $\rightarrow$ F)))	-->	[every].	5
noun(X, human(X))	-->	[human].	6
noun(X, mortal(X))	-->	[mortal].	7
aux_verb	-->	[is].	8

We notice from the above expression that the determiner *every* of noun phrase builds the logical form of entire sentence as $all(X, LF)$, where LF is a logical form of the type $(Base \rightarrow Focus)$, where Base is a logical form of the head noun of noun phrase, e.g., $human(X)$ and Focus is a logical form determined by verb phrase likes a mortal(X).

Similarly, we can write DCG rules for handling LF for sentences such as 'every woman likes a child' that can return LF as ' $all(X, (woman(X) \rightarrow exist(Y, (child(Y) \wedge likes(X, Y))))$ '. The following DGC rules handle such type of sentences are given in Table 16.26.

Table 16.26 DCG for Generating LF (Sentence Format: Every Noun likes/loves a Noun)

DCG Rules			Rule Number
sentence(LF)	-->	np(X, F, LF), vp(X, F).	1
np(X, F, LF)	-->	det(X, B, F, LF), noun(X, B).	2
vp(X, F)	-->	verb(X, Y, V), np(Y, V, F).	3
det(X, B, F, all(X, (B $\rightarrow$ F)))	-->	[every].	4
det(X, B, F, exist(X, (B $\wedge$ F)))	-->	[a].	5
noun(X, woman(X))	-->	[woman].	6
noun(X, child(X))	-->	[child].	7
verb(X, Y, likes(X, Y))	-->	[likes].	8

Now we execute the following goal with respect to Prolog program (Table 16.26) and display the corresponding trace in Table 16.27.

Query: ?- sentence(LF, [every, woman likes, a, child], []).

Table 16.27 Trace of Goal ?- Sentence (LF, [every, woman, likes, a child], [])

Bindings	Trace	Rule applied
	?- sentence(LF, [every, woman, likes, a, child], []).	1
	?- np(X, F, LF, [every, woman, likes, a, child], Z), vp(X, F, Z, []).	2
LF = all(X, (B $\rightarrow$ F)), Z1 = [woman, likes, a, child]	?- det(X, B, F, LF, [every, woman, likes, a, child], Z1), noun(X, B, Z1, Z), vp(X, F, Z, []).	4
B = woman(X), Z = [likes, a, child]	?- noun(X, B, [woman, likes, a, child], Z), vp(X, F, Z, []).	6
V = likes(X, Y), Z = [a, child]	?- vp(X, F, [likes, a, child], []). ?- verb(X, Y, V, [likes, a, child], Z), np(Y, V, F, Z, []).	3
	?- np(Y, likes(X, Y), F, [a, child], []).	8
F = exist(Y, (B $\wedge$ likes(X, Y)), Z = [child]	?- det(Y, B, likes(X, Y), F, [a, child], Z), noun(Y, B, Z, []).	5
B = child(Y)	?- noun(Y, B, [child], []).	7
Succeeds LF = all(X, (woman(X) $\rightarrow$ exist(Y, (child(Y) $\wedge$ likes(X, Y)))))		

On backtracking construct the result as follows:

F	=	exist(Y, (child(Y) L likes(X, Y)))
B	=	woman(X)
LF	=	all(X, (B $\rightarrow$ F))
Therefore, LF	=	all(X, (woman(X) $\rightarrow$ exist(Y, (child(Y) $\wedge$ likes(X, Y)))))

We have shown three sets of grammar rules (Tables 16.24, 16.25, 16.26) to explain the concept of rule formulation using semantic arguments for different types of sentences. A general grammar handling all types of sentences could be formed by combining all the rules carefully.

16.5.3 Case Grammar

Case grammar has been introduced in earlier chapter on Advanced Knowledge Representation Techniques. The case structure may vary depending on the kind of applications developed. Some of the applications are mentioned below where case frame structure may be used as a semantic representation.

- *Paragraph Comprehension*: It is used for answering implicit questions. Here we need to focus on deep cases.
- *General question answering system*: It will require surface knowledge of the sentences to be stored along with deep cases.
- *Machine Translation System*: It requires target sentences to be formulated. For this we require details of surface structure knowledge also to be stored. So, we need lot many case slots to capture syntactic as well as semantic information.

Case Frame Representation for Machine Translation

We developed Machine Translation System across three different languages (English, Hindi and Telugu) using case frame as intermediate language as a part of Media Lab Asia sponsored project in 2006. We chose case frame representation as an intermediate representation. Developing pairwise machine translation systems between N languages, would require N^2 translators, whereas using case frame (or any interlingua representation), $2*N$ modules are required; N for converting source language to case frame and N modules for generating from case frame to sentences of target language. We need to write rules for converting from the source language to case frame and from case frame to the target language. The sentences for translation were restricted to all three tenses with indefinite and continuous forms for English language. Case frame is represented using English language irrespective of source or target languages for the sake of convenience. Each language dictionary contained case-word (English) and corresponding language word (e.g., man(case word) – aadami(Hindi)). Nouns of any language in the dictionary had various syntactic and semantic attributes attached to it. For example, attributes such as animate, non-animate, locatable, eatable, etc. With this case frame framework as an intermediate representation, we have translated source language sentence into case frame structure and from case frame to any source