Chapter 19
Natural Language Processing

Abstract The abundant volume of natural language text in the connected world, though having a large content of knowledge, but it is becoming increasingly difficult to disseminate it by a human to discover the knowledge/wisdom in it, specifically within any given time limits. The automated NLP is aimed to do this job effectively and with accuracy, like a human does it (for a limited of amount text). This chapter presents the challenges of NLP, progress so far made in this field, NLP applications, components of NLP, and grammar of English language—the way machine requires it. In addition, covers the specific areas like probabilistic parsing, ambiguities and their resolution, information extraction, discourse analysis, NL question-answering, commonsense interfaces, commonsense thinking and reasoning, causal-diversity, and various tools for NLP. Finally, the chapter summary, and a set of relevant exercises are presented.

Keywords Natural language processing · Challenges of NLP · Natural language parsing · Probabilistic parsing · NL ambiguities · Ambiguity resolution · Commonsense interfaces · Commonsense thinking · Commonsense reasoning · Discourse analysis · Question-answering · Causal-diversity · NLP tools

19.1 Introduction

A language is a set of finite length sentences, constructed using a finite alphabet set, or in terms of language syntax, they are constructed using a finite vocabulary of symbols. Since the alphabet set is finite, as well as the length of the sentences, the set of sentences (in a language) is also finite. For example, if alphabet set is of size two, and length of sentences is ten, there can be only 1024 maximum number of sentences possible. However, in many of our theoretical studies we consider the language as infinite, hence size of some sentences may also be infinitely large. For our study of languages, we usually consider the languages without any bound, i.e., infinite, but for specific languages, which are of practical use, we limit our study to finite set. This is because, the machines we would like to use for processing the

K. R. Chowdhary, *Fundamentals of Artificial Intelligence*,
https://doi.org/10.1007/978-81-322-3972-7_19

languages, are finite machines, they have finite memory, and have finite processing power.

When we are interested in an infinite language set, say L, we can use the finite devices called *generating grammars* to investigate the structure of L. In that scenario, the theory of language will contain specification of a class of functions from which grammars for a particular language may be drawn. Natural language processing (NLP) is a collection of computational techniques for automatic analysis and representation of human languages, motivated by theory. However, the automatic analysis of text, at par with humans, requires a far deeper understanding of natural language by machines, which is still far from reality. There are many examples of NLP, like, online information retrieval, aggregation, and question-answering, have been mainly based on algorithms relying on the textual representation of web pages, as well NLP to some extent. Such algorithms are very good at retrieving texts (IR), splitting it into parts, checking the spellings, and and word-level analysis, but not successful for analysis at sentence and paragraph level. Hence, when it comes to the question of interpreting sentences and extracting meaningful information, however, the capabilities of these algorithms are still very limited.

NLP, in general, requires high-level symbolic capabilities, which includes the following:

Access and acquisition of lexical, semantic, and episodic characteristics,
Creation and propagation of dynamic bindings,
Manipulation of constituent recursive structures,
Coordination of many processing and learning modules,
Identification of basic language constructs (e.g., objects and actions) and,
Representation of abstract concepts.

All the above capabilities are needed to shift from mere NLP to what is usually called as natural language understanding. The present approaches to NLP are based on the syntactic representation (also called syntactic structures) of text, i.e., relying on the word co-occurrence frequencies. Such algorithms have the limitation that they can process only based on the information they can see in the text being processed, but cannot consider the background information what we humans do. For example, when we say that "Sachin Tendulkar is a good batsman," we understand this sentence due to abundant information we have in our brain about the game of cricket, and bout the successes of "Sachin Tendulkar" in many cricket matches. However, the present algorithms do not have all this background with them, hence their understanding is limited about any given text.

As human text processors, being humans we do not have such limitations, as every word coming across in the text activates a cascade of semantically related concepts, relevant episodes, and sensory experiences, all these enable the completion of complex NLP tasks. These tasks are, for example, word sense disambiguation, textual entailment, and semantic role labeling, all in a quick and effortless way.

Many new computational models are making attempts to bridge the cognitive gap by emulating the processes recognized as being part of the human brain, and used for language processing by humans. These approaches depend on semantic features

that cannot be explicitly expressed through the text. The computational models are useful for theoretical purposes, e.g., scientific studies, such as exploring the nature of linguistic communication and its properties, as well for practical and industrial applications, such as enabling effective human–machine communications.

Challenges of NLP

Developing a program that understands natural language is a difficult problem. Majority of natural languages are large, they contain infinitely many sentences. Also there is much ambiguity in natural language. Many words have several meanings, such as *can*, *bear*, *fly*, *orange*, and the same sentences many times have different meanings in different contexts. Due to this, creation of programs that understands a natural language is a challenging task.

The syntax of a language helps us to decide how the words are being combined to make larger meanings. For example, in the sentence "the dealer sold the merchant a dog," it is important to be clear about what is sold to whom. Some of the common examples are the following:

I saw the Golden gate bridge flying into San Francisco.
(Is the bridge flying?)
I ate dinner with a friend.
I ate dinner with a fork.
Can companies litter the environment
(Is this a statement or question?)

Finally, assuming that we have overcome the problem at the previous levels, we must create an internal representation, and then, some how use the information in an appropriate way. This is the level of *semantics* and *pragmatics*. Here too the ambiguity is prevalent. Consider the following sentences.

Jack went to store. He found the milk in aisle three. He paid for it and left.

Here the problem is deciding whether "it" in the sentence refers to aisle or three, the milk, or even the store.

The most important part in the above is what is internal representation, so that these ambiguities in understanding the sentence do not occur and machine understands the way a human being understands the sentences.

Learning Outcomes of this Chapter:

1. Define and contrast deterministic and stochastic grammars, providing examples to show the adequacy of each. [Assessment]
2. Simulate, apply, or implement classic and stochastic algorithms for parsing natural language. [Usage]
3. Identify the challenges of representing meaning. [Familiarity]
4. List the advantages of using standard corpora. Identify examples of current corpora for a variety of NLP tasks. [Familiarity]

5. Identify techniques for information retrieval, language translation, and text classification. [Familiarity]
6. Tools for NLP. [Usage]

19.2 Progress in NLP

Right from its start, NLP focused on these tasks: natural language translation, information retrieval, information extraction, text summarization, question answering, topic modeling, and the recent one on opinion mining.

Syntax Versus Semantics

The NLP advances that took place after the 1960s paid attention on syntax, and partly on the semantics, but it was more of syntax-driven processing [1]. However, the semantic problems of Natural Language and its processing were obvious requirements from the beginning itself, but strategy adopted was to tackle the syntax first, for the more direct applications. Many works on NLP recognized the need for external knowledge for interpretation and reasoning on input language (Minsky, 1968), who explicitly emphasized semantics, e.g., the general-purpose semantics with case structures for representation and semantically driven processing [17].

First-Order Predicate

One of the most popular Natural Language representation technique used is FOPL (First-Order Predicate Logic), which is a deduction-based logic consisting of *axioms* and *inference* rules. The FOPL can be used to construct relationally rich predicates formulas, using quantification. The inferences can be drawn using formulas, called knowledge base, with the application of *resolution principle*. The FOPL supports the expressions of syntactic, semantic and, to a certain extent pragmatic. The syntax expresses a way of grouping of symbols, so that they are considered properly formed. The semantics specifies the meaning of these well-formed expressions. The pragmatics is more difficult to implement, which specifies to make use of context-related information to provide better correlations between different semantics. The later is important in tasks like word sense disambiguation.

Default Logic

The Default Logic was proposed [15] to formalize the default assumptions, e.g., "all birds fly." However, problems arose when default logic formalized the facts that were true in the majority of cases but false in the exceptional cases, i.e., "exceptions to these general rules," e.g.' "penguins do not fly" [15].

Production Rules

The *production rules* based model is another popular model for representation of Natural Languages (Chomsky, 1956). This model keeps ongoing assertions in a volatile working memory. The production rules comprise as their premises the set of conditions and consequent as a set of actions (i.e., IF <conditions> THEN <actions>).

The basic operations in a production-based system comprise, in order, three steps: 1. Recognize, 2. Resolve the conflict, and 3. Act. These three operations repeat in order as cycle until there are no more rules left in the working memory on which these rules can be applied. The first step (recognize) identified the set of rules (called conflict set) whose premises conditions are satisfied by the current working memory. The second step (resolve conflict) looks into the conflict set and selects a set of suitable rules to execute. The suitability is with the objective of efficiency, such that the task can be completed in shortest time. The "act" step is the third step, which simply executes the actions and updates the working memory. The production rules are created in modular forms for ease of writing as well for optimum use of memory and processing requirements.

OWL

A new model for NLP is Ontology Web Language (OWL), which is an XML-based vocabulary that extends the Resource Description Framework (RDF) language and provides a more comprehensive set for representation of ontology. Some examples of ontology representations are: definition of classes and relations between them, properties of the classes, and constraints on relations and on properties of the classes. The subject–predicate–object model is supported by RDF, which makes assertions about the resource. The RDF-based engines are commonly used for checking semantic inconsistency, which helps to improve on the ontology classification.

Networks

The networks as tools have been commonly used for NLP, where inferencing is used to be the primary goal. For example, the Bayesian belief networks (Pearl, 1985) is the tool for expressing joint probability distribution over many interrelated hypotheses. The variable in such networks are represented using directed acyclic graph (DAG), the edges are causal connections between any two variables, such that the truth of the causing variable directly influences the caused variable. A Bayesian network represents the subjective degree of confidence. The representation explores the role of *prior knowledge* and combines the pieces of evidence of the likelihood of the events.

For computation of joint probability distribution of the belief network, we need to follow the following steps: for each variable q represented through belief network, there is a need to know the probability $P(q|p_1, ..., p_n)$, where $p_1, ..., p_n$ are the parents of q. In a large network it requires such computations at every node, making it difficult to determine the joint probability distribution of the entire network. It will require maintenance of probability table at each node in the belief network, which is again a challenging task for large-scale information processing problems.

Finally, the Bayesian networks also have limited expressiveness in NLP, as it is no better than that of proposition logic. Due to these factors, semantic networks are more often used in NLP [13].

Semantics Networks

A semantic network is a graphical notation for representing using interconnected nodes and arcs (Sowa, 1987), which is quite different from belief networks. The

network's focus is on relationships between a concept and a newly defined sub-type, represented using *Isa* keyword. Such a network structure provides generalization, which supports the rule of *inheritance*. The latter has the property of the copying the properties defined for super-type to all the sub-types, there is no need to define the properties again for the sub-types. The information represented through the semantic networks is assumed to be true.

Apart from the inheritance-based networks, another kind of semantic networks is the *assertional network*. This network is meant to assert propositions, and the information it contains is assumed to be contingently true. However, the truth of contingent is not implemented through default logic, but it is based on application of human's commonsense. The proposition also has sufficient reason, such that the reason entails the proposition, e.g., "the rock is hot" with the sufficient reasons being "the sun is shining on the rock" and "whatever the sun shines on is warm" [19].

The concept of semantic network came in the early 1960s, which was further developed by Marvin Minsky, till the 1980s. The base of semantic network is human mind, where it is hypothesized that the human intelligence results from a vast majority of things, and not from any single perfect principle. As per Minsky, the human mind (or even animals') is made of many little things, which he termed as "agents", but each one of them is mindless, when considered as an independent element. But, these agents lead to true intelligence when they work collectively, and responsible for performing functions, like, remembering, generalizing, comparing, analogizing, exemplifying, predicting, simplifying, etc.

The theory of human cognition by Minsky gave birth to many research projects' attempts, that built commonsense knowledge bases for NLP tasks. The major representative projects related to cognition are as follows.

Cyc—a Large Knowledge Based System [6],
WordNet—a Large Knowledge Based System [8],
Thought-Treasure—Natural Language Processing with Thought-Treasure [11], and
Open Mind Common Sense project—a second-generation commonsense database [18].

19.3 Applications of NLP

The importance of NLP is due to the fact that there is a huge amount of data in WWW, at least 20 billion pages, and that can be used as a big resource, provided that important information can be found from these through NLP. Some of the well-known uses of this data are possible through the following applications [4]:

Indexing and searching large texts,
Information retrieval (IR),
Classification of text into categories,
Information extraction (IE),
Automatic language translation,

Automatic summarization of texts,
Question-answering (QA),
Knowledge acquisition,
Text generations/dialogues.

Some of the above domains of NLP will be discussed in this chapter, and we will get the exposure sufficient enough to solve many problems of NLP however, the field of NLP is so vast that even a book is not sufficient to present in detail all the available techniques.

19.4 Components of Natural Language Processing

The NLP is the subject of computational linguistics—the study of computer systems for understanding and generating natural language. The linguistics has two branches—computational linguistics and theoretical linguistics. The computational linguistics has been concerned with developing algorithms for handling a useful range of natural language as input. While the theoretical linguistics has focused primarily on one aspect of language performance, grammatical competence—how people accept some sentences as correctly following grammatical rules and others as ungrammatical. They are concerned with language universals—principals of grammar which apply to all natural languages [16].

Computational linguistics is concerned with the study of natural language analysis and language generation. Further, the language analysis is divided into two domains, namely sentence analysis, and discourse and dialogue structure. Much more is known about the processing of individual sentences than about the determination of discourse structure. Any analysis of discourse structure requires a prerequisite as an analysis of the meaning of individual sentences. However, it is a fact that for many applications, thorough analysis of discourse is not mandatory, and the sentences can be understood without that [14].

The sentence analysis is further divided into *syntax analysis* and *semantic analysis*. The overall objective of sentence analysis is to determine what a sentence "means". In practice, this involves translating the natural language input into a language with simple semantics, for example, *formal logic*, or into a *database command language*. In most systems, the first stage is syntax analysis. Figure 19.1 shows the relations among different components of NLP.

19.4.1 Syntax Analysis

The syntax analysis of a language performs two main functions in analyzing natural language text, which are as follows:

- *Determining Structure.* Given the sentences, the syntax analysis should identify the *subject* and *object* of each verb, and determine what each modifying word is,

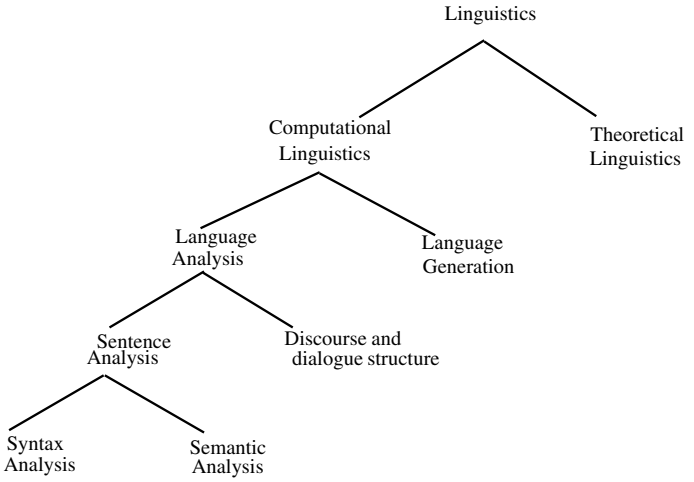

Fig. 19.1 Components of NLP

and what phrase it modifies. This is achieved by assigning a tree structure to the input, during the process called as *parsing*.

- *Regularizing the syntax structure.* Subsequence processing, i.e., semantic analysis gets simplified if a large number of possible input sentences are mapped into a small number of structures. For example, some material in sentences (enclosed in brackets in the example below) can be omitted, but still maintaining the meaning of these sentences unaltered.

  "He talks faster than John [talks]."
  "John ate cake and Mary [ate] Cookies."

In addition, if the structures are appropriately chosen, operator–operand (e.g., verb–subject, verb–object) relations should be clearly evident in the output of the syntactic stage. This kind of regularization is achieved, when passive voice sentence is converted into active voice, for example.

"Those grapes were crushed by me,"
converted to
"I crushed those grapes."

Some NLP type have chosen to omit syntactic regularization all together, and have semantic component operate directly on the full variety of sentence structures. In such systems, the syntactic regularization is subsumed within the semantic analysis process, which will, however, require more complex semantic rules.

The syntax analysis is performed through grammars (phrase structure grammar) using the process of parsing, and through transformational grammars, discussed in the following sections.

19.4.2 Semantic Analysis

The objective of semantic analysis is to determine what a sentence means. The semantics seeks the conditions under which the sentence is true. Almost equivalently, we can say, what are the inference rules for the sentences of the language. Characterizing the semantics of interrogative and imperative sentences are more problematic.

Why the meaning of a sentence is important as long as a sentence is correct or syntactically correct? Here is one reason. Consider that we build a natural language system, our aim is not only to ensure that sentence is correct, but usually we want the system to do some thing in response to input, like, retrieve data, move a robot arm, etc. In general, this will mean translating the natural language input into the formal language of a database retrieval system, a robot command system, etc. In comparison with natural language, these formal languages will have the following properties:

- unambiguous,
- simple rules of interpretation and inference, and
- logical structure determined by the form of the sentence.

Thus, there is a significant similarity between the tasks that translate into such a *practical language*, and those translate into a *logical formalism*.

Formal Languages for Meaning Representation

One method for meaning representation is *propositional Logic*. Using this logic, given that "If it is raining then ground is wet, and It is raining," we can infer that "ground is wet." We observe that as long as we preserve the general structure of the argument, including the "if ... then ..." connective in the first part, we may change the sentences making up the argument and still have a valid argument. For example, "If it is night then Ram is sleeping, and It is night," we may conclude that "Ram is sleeping." And, the overall argument is a valid argument.

The above rule is represented in predicate logic as

Given that: $P \rightarrow Q$, and
P,
we may conclude: Q.

The above rule is called *Modus Ponens*. Similar to propositional logic, the rule of Modus Ponens also exists in *Predicate Logic* also. For details of these logics, refer to chapters two and three in this book.

The other approaches for semantics are *Semantic networks*, and *Conceptual dependency*, which are presented in details in the chapter seven of this book.

19.4.3 Discourse Analysis

So far we have discussed the structure and meaning of individual sentences, but what is solution is the meaning of the entire text. Because, the information conveyed by a

text is clearly more than sum of its parts—more than the meaning of its individual sentences. For example, if a text tells a story, describes a procedure, or presents an argument, we must understand the connections between the component sentences in order to have fully understood the story. *Discourse analysis* is the study of these connections between the sentences. Since these connections are implicit in the text, identifying them may be a difficult task.

As a simple example of the challenge of discourse analysis, just consider the following brief description of a naval encounter:

> Just before the dawn, the *Vikrant* sighted the an unidentifiable blue-ship and fired two torpedoes. It sank swiftly, leaving one survivor.

The problem we face is, what stands for "it"? There are four candidates in the first sentence: dawn, Vikrant, blue-ship, and torpedo. The semantic analysis helps to exclude the "dawn" as a meaning for "it", and number agreement excludes "torpedoes". But, still leaves two candidates Vikrant, and blue-Ship, which are both ships. Our syntax and semantic analysis tools will not enable us to resolve between these two, as which stands for "it". To get the noun for "it", we must understand the causal relationship between the firing of torpedoes and sinking of ship. Since Vikrant fired torpedoes, it must have fired on the blue-ship (because Vikrant is not supposed to fire on itself !). Hence, "it" stands for blue-ship.

Since much of the information conveyed by this text is implicit, we cannot claim that the text is adequately understood unless the implicit information is recovered. The role of discourse analysis is to recover this information.

As we have discussed that discourse is a multi-sentence text, discourse comes in many forms, describe a scene, give instructions, or present an argument. The tools for discourse analysis are *Frames*, *Scripts*, and *Conceptual dependencies*, which were covered in detail in chapter seven in this book.

19.5 Grammars

The grammar of a language can be viewed as a theory of the structure of that language. Like any mathematical theory, the theory of grammar is based on a a set of finite number of observed sentences, but it projects this to an infinite set of grammatical sentences. This becomes possible by creating general laws (grammatical rules), framed in terms of such hypothetical constructs as the particular phonemes, words, and phrases, of the spoken language under analysis. A properly formulated grammar should be able to determine unambiguously the set of all grammatical sentences in that language [2].

Let us assume that for most languages there are certain clear examples of grammatical and ungrammatical sentences. Consider the following sentences of in English:

1. John ate a sandwich,
2. Sandwich ate a John.

Suppose a large corpus-x does not contain either of these sentences. In this scenario can we say, grammar that is determined for corpus-x will direct the corpus to include the first sentence and exclude second. Even though such simple cases maybe not adequate to confirm that all correct sentences of that language will be tested as correct and all wrongs will be rejected, but still it becomes a strong test.

The first step toward the linguistic analysis of a language is to provide a *finite system* of representation for its sentences. By the grammar of a language L we mean a device that produces all of the strings that are sentences of L and no other strings.

19.5.1 Phrase Structure

The description of phrase structure comprises the grouping of words into phrases, which are then grouped into smaller constituent phrases and so on, until the ultimate constituents (generally morphemes) are reached. The phrases are typically classified as *noun phrases* (NP), *verb phrases* (VP), etc. For example, the sentence "the man took the book," might be analyzed as shown in Fig. 19.2.

Evidently, description of sentences in such manner permit considerable simplification compared tor the word-by-word model. This is because the composition of a complex class of expressions such as NP, can be stated just once in the grammar, and this class can be used as a building block at various stages in the construction of sentences. We now question, what form of grammar corresponds to this conception of linguistic structure? We are going to get the answer in the following.

19.5.2 Phrase Structure Grammars

A phrase structure grammar is defined using a finite vocabulary (alphabet) Σ, a finite set V of variables, a finite set P (rewrite rules) of productions of the form $X \rightarrow Y$, where X is a string in V, and Y in $V \cup \Sigma$. Hence, a grammar is $G = (V, \Sigma, S, P)$, where Σ is set of terminal symbols, which appear at the end of generation, S is start symbol (for sentence). The corresponding language of G is $L(G)$. Given this grammar, we say that a string $\beta \in (V \cup \Sigma)^*$ follows from $\alpha \in (V \cup \Sigma)^*$, then we write it as $\alpha \rightarrow \beta$. Also, if we are able to generate a sentence w from start symbol

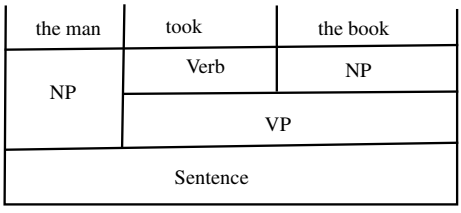

Fig. 19.2 A sentence's structure

S, then we can write it as

$$S \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n = w. \tag{19.1}$$

A derivation of a sentence is thus roughly analogous to a proof with V taken as axiom system and P as the rules of inferences. We say that L is a *derivable language* if L is the set of strings that are derivable from grammar G, and say that L is a terminal language if it is the set of terminal strings from some system (V, Σ, S, P).

As a simple example of a system of this form, consider the following small part of English grammar:

$$\begin{aligned} S &\to NP\ \ VP \\ VP &\to Verb\ \ NP \\ NP &\to the\ \ man \mid the\ \ book \\ Verb &\to took \end{aligned} \tag{19.2}$$

Among the derivations from (19.2) we have, in particular:

$$\begin{aligned} D_1 : S &\to NP\ \ VP \\ &\to NP\ \ Verb\ \ NP \\ &\to the\ \ man\ \ Verb\ \ NP \\ &\to the\ \ man\ \ took\ \ NP \\ &\to the\ \ man\ \ took\ \ the\ \ book \end{aligned}$$

Also, we have:

$$\begin{aligned} D_2 : S &\to NP\ \ VP \\ &\to NP\ \ Verb\ \ NP \\ &\to the\ \ man\ \ Verb\ \ NP \\ &\to the\ \ man\ \ Verb\ \ the\ \ book \\ &\to the\ \ man\ \ took\ \ the\ \ book. \end{aligned}$$

These derivations D_1 and D_2 are evidently equivalent; they differ only in the order in which the rules are applied. We can represent this equivalence graphically by constructing diagrams that correspond, in an obvious way, to derivations. Both D_1 and D_2 reduce to the diagram as shown in Fig. 19.3. The diagram gives the phrase structure of terminal sentence "the man took the book." Note that "the man", "took", "the book", and "took the book" are phrases as these are traceable back to some nodes. However, "man took" is not a phrase as it is not traceable back to any node.

Example 19.1 Consider that various tuples of a grammar are given as follows:

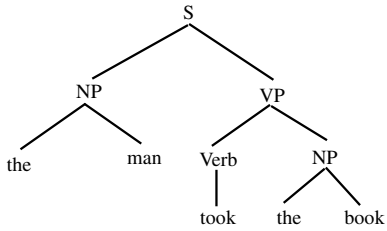

Fig. 19.3 Phrase structure of "the man took the book"

$$
\begin{aligned}
V &= \{S, NP, N, VP, V, Art\} \\
\Sigma &= \{boy, icecream, dog, bites, likes, ate, the, a\} \\
P = \{S &\rightarrow NP \;\; VP, \\
NP &\rightarrow N, \\
NP &\rightarrow Art \;\; N, \\
VP &\rightarrow V \;\; NP, \\
N &\rightarrow boy \mid icecream \mid dog, \\
V &\rightarrow ate \mid likes \mid bites, \\
Art &\rightarrow the \mid a\}
\end{aligned}
$$

Using the above grammar, we can generate the following sentences:

The dog bites boy.
Boy bites the dog.
Boy ate Icecream.
The dog bites the boy.

Note that, to generate any sentence, rules from the set P are applied sequentially starting from the first rule. However, a grammar does not guarantee the generation of meaningful sentences, but generate only those that are structurally correct as per the rules of the grammar. □

Structural Representation

It is convenient to represent the sentences as tree or a graph to help expose the structure of the constituent parts, called parts-of-speech (POS). For example, the sentence, "The boy ate an Icecream" can be represented as a tree shown in Fig. 19.4.

For the purpose of computation, a tree must also be represented as a record, or a list, or some similar data structure. For example, the tree above in Fig. 19.4 can be represented as a list:

```
(S (NP ((Art the)
        (N boy))
   (VP (V ate) (NP (Art an) (N Icecream)))))
```

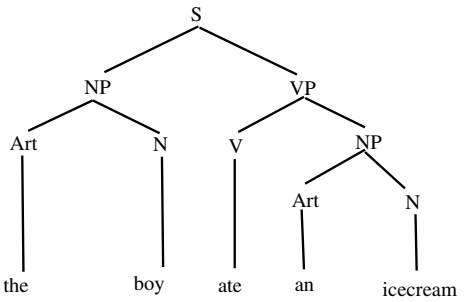

Fig. 19.4 A Syntax Tree of "The boy ate an Icecream"

A more extensive English grammar can be obtained with the addition of other constituencies such as prepositional phrases (PP), prepositions (Prep), adjectives (ADJ), determiners (DET), adverbs (ADV), auxiliary verbs (AUX), and many other features. Correspondingly, the additional rewrite rules are as follows:

$$PP \rightarrow Prep\ \ NP$$
$$VP \rightarrow V\ \ ADV \mid V\ \ PP \mid V\ \ NP\ \ PP \mid AUX\ \ V\ \ NP$$
$$Det \rightarrow Art\ \ ADJ \mid Art$$

These extensions allow the increase in complexity of the sentences generated through this expanded grammar, along with its expression power, as in the following sentences.

The(Art) cruel(Adj) man(N) locked(V) the(Art) dog(N) in(prep) the(Art) cage(N).
The(Art) laborious(Adj) man(N) worked(V) to(Aux) make(V) extra(Adj) money(N).

19.6 Classification of Grammars

The grammars are classified in many ways: when they are based on their generating rules they are called *Chomskian grammars*; when a grammar transforms the passive-voice sentences to active voice, it is *transformational grammar*; and when a sentence generated by a grammar has more than one meaning(sense), it is called *ambiguous grammar*. The following describes the theory of each in brief.

19.6.1 Chomsky Hierarchy of Grammars

In fact, it is not always possible to formally characterize the natural languages with a simple grammar as discussed above. The grammars are defined by *Chomsky Hierarchy*, as type 0, 1, 2, 3, with simplest as type-3 and most complex as type-0. The

type-3, called *Regular Grammars* have rewrite rules as:

$$S \rightarrow aS$$
$$S \rightarrow a.$$

We note that the left-hand side is a variable symbol, and right hand is only terminal symbol, or a terminal followed with variable symbol.

The type-2 grammars, called *context-free grammars* (CFG) have left-hand side symbol of variable, and right-hand side has any combination of variables and terminal symbols. Following all the productions belong to type-2 grammars, where S, A, B are variable symbols; a, b are terminals, and α is any combination of variables and terminals.

$$S \rightarrow aS \mid aSb \mid aB \mid aAB \mid \alpha$$
$$A \rightarrow a$$
$$B \rightarrow b$$

The typical rewrite rules for type-0, called *unrestricted grammar* and type-1 called context-sensitive grammar (CSG), are are follows:

$$S \rightarrow aS \mid aAB$$
$$AB \rightarrow BA$$
$$aA \rightarrow ab \mid aa$$
$$\alpha \rightarrow \beta$$

In the above rules, $\alpha \in (\Sigma \cup V)^* \cup V$, and $\beta \in (\Sigma \cup V)^*$. In all these rules, left-hand sides are both variables and terminals and same applies for right-hand symbols. The only difference between type-0 and type-1 grammars is that, if $\alpha \rightarrow \beta$ is production, then in type-1, $|\alpha| \leq |\beta|$.

Since, the grammars are called unrestricted, context-sensitive, context-free, and regular grammars, the corresponding languages are also called unrestricted, context-sensitive, context-free, and regular languages, respectively. The natural languages are mostly based on the type-2 languages, as the type-0 and type-1 are not much understood yet, and are difficult to implement.

19.6.2 Transformational Grammars

Transformational Grammar involves the use of defined operations called transformations, to produce new sentences from existing ones. The grammars discussed above

produce different structures for different sentences, even though they have the same meaning. For example,

Ram gave Shyam a book.
A book was given by Ram to Shyam.

In the above, the subject and object roles in second sentence are switched. In the first, subject is "Ram" and object is "Book", while in second sentence they are other way round. This is an undesirable feature for machine processing of a language. In fact, sentences having the same meaning should map to the same internal structures.

However, by adding some extra components, we can produce a single representation for sentences having the same meaning, through a series of transformations. This extended grammar is called *Transformational grammar*. In addition, *semantic* and *phonological* components are components, which are added as new, helps in interpreting the output of the syntactic components, as meaning and sound sequences. These transformations are tree manipulation rules, which are taken from dictionary, where words contain semantic featuring each of the lexica.

Using transformational generative grammar, a sentence is analyzed in two stages: (1) basic structure of the sentence is analyzed to determine the grammatical constitutional parts, which provides the structure of the sentence. (2) This structure is transformed into another form, where a deeper semantic structure is determined.

The application of transformations is to produce a change from passive voice form of the sentence into active voice, which changes a question to declarative form, handle negations, and provide subject–verb agreement. Figure 19.5a–c shows the three stages of conversion, from passive voice to active voice of a sentence.

However, the transformational grammars are now rarely used as computational models for language processing [14].

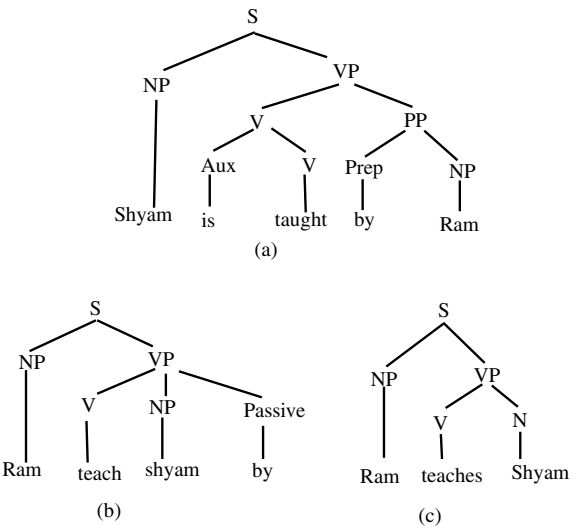

Fig. 19.5 Transformational Grammar

19.6.3 Ambiguous Grammars

An ambiguous grammar is a context-free grammar for which there exists a sentence that can have more than one parse tree, while an unambiguous grammar is a context-free grammar for which every sentence has a unique parse-tree. For computer programming languages, the reference grammar is often ambiguous, due to issues such as the dangling "Else" problem. If present, some of the ambiguities can be resolved by adding precedence rules or other context-sensitive parsing rules, so the overall phrase grammar is unambiguous. Note that, to find out if general grammar is ambiguous, is a NP-hard problem [4].

The ambiguous grammars have more than one parse-tree for the same sentence. For example, consider the sentence "He drove down the hill in the Car." A process for constructing a parse-tree is grouping the words to realize the structure in the sentence. The parse-tree given in Fig. 19.6a for this sentence shows the grouping of words and Fig. 19.6b shows the parse-tree for this sentence. Similarly, Fig. 19.7a

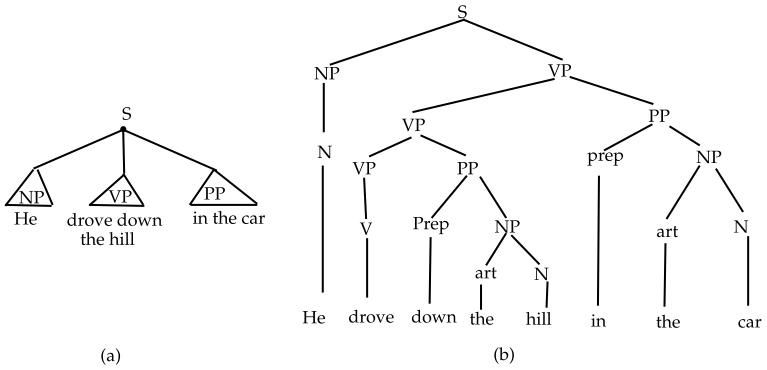

Fig. 19.6 **a** Grouping the words, **b** Parse-tree I: "He drove down the hill in the car"

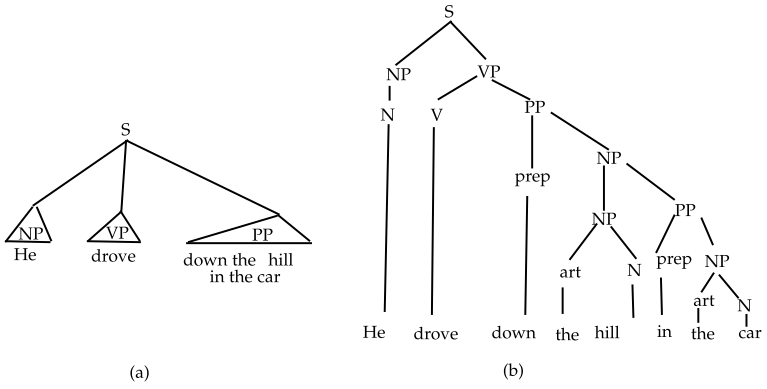

Fig. 19.7 **a** Grouping the words, **b** Parse-tree II: "He drove down the hill in the car"

shows another grouping for the same sentence and the corresponding parse-tree is shown in Fig. 19.7b. Since this sentence can be generated through two different parse-trees, the sentence is ambiguous, and hence its grammar is also ambiguous.

19.7 Prepositions in Applications

Prepositions, as well as prepositional phrases (PPs) are important to the precise understanding of any language, which may be useful for certain applications, e.g., in Information Extraction. However, they have been ignored on the grounds of being syntactically promiscuous, and semantically having no significant meaning, hence they were considered in the rank of "stop words" [20].

As a part of its argument structure, property of a preposition being subcategorized or specified by the governor is called *selection* property. The governor is usually a verb. An example of selection preposition is, "with", like, in the phrase, *dispense* with *introductions*. Here *dispense* is verb and *introductions* is the object of this verb. But this object is realized in a prepositional phrase, which is headed by *with*. Conventionally, the prepositions of *selection* type are uniquely specified. The following are other examples with well-defined structures.

$chuckle over/at$

It is clear from the above examples the use of prepositions have no definite semantics.

The selection proposition is useful in NLP applications; it operates at the *syntax-semantics* interface, i.e., those application which openly translates surface strings onto semantic representations, or vice versa. Examples of such translation are: Information Extraction, and Machine Translation using some form of interlingua.

PP Attachment

To find a PP attachment is nothing but to find the governor for a given PP. Consider the sentence,

$Sujataeats^{1}kheerwithspoon.$

This sentence[1] consists a syntactic ambiguity, as in one case the PP "with spoon" is governed by noun *kheer* (i.e., as part of the NP *kheer with spoon* (see Fig. 19.8a). As an alternative sense the PP "with spoon", is governed by the verb *eats* (i.e., as a modifier of the verb, as indicated in Fig. 19.8b. Of these, the later case of verb attachment (i.e., Fig. 19.8b) is, of course, the correct analysis.

The much interest in PP *attachment* comes from the PP being a common phenomenon when parsing the languages such as English, which is a major cause of parser errors. The ambiguity due to PP is called *attachment ambiguity*.

The syntactic preferences do not provide the solutions in dealing with PP attachment, and they are not very effective for predicting the difference in PP attachment

[1] Kheer: A sweet dish, like porridge, popular among the Indians.

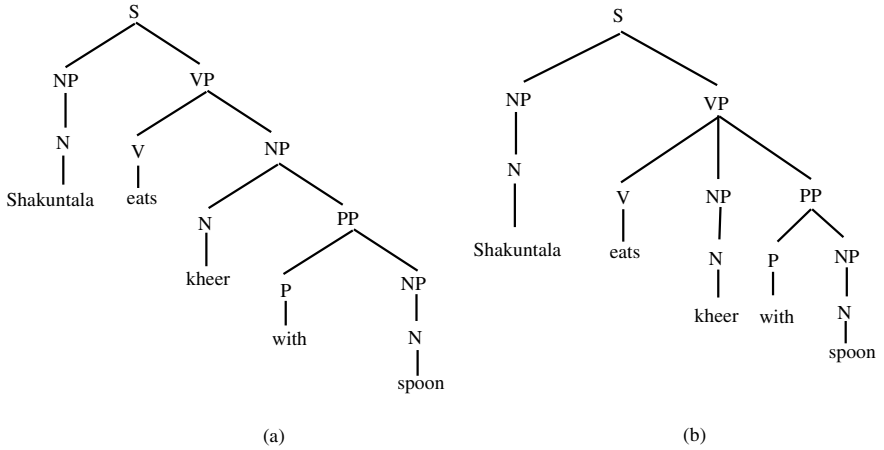

Fig. 19.8 The prepositional phrase "with spoon," being governed by: **a** noun *kheer*, **b** verb *eats*

between noun attachment and verb attachment. This difficulty led to a shift from syntactic approaches in the 1980s toward AI-based approaches, the later used world knowledge for resolving PP attachment ambiguity. Figure 19.8a shows that, due to the availability of the knowledge that *spoon* is an eating implement, it would suggest a preference for verb attachment. In a similar way, the knowledge that they are not a foodstuff, it would suggest a preference against noun attachment.

An algorithm that resolves the attachment ambiguity, defines constraints that are derived from *semantic* and world knowledge. These constraints are used for composing the meaning of the child unit, which is attached to the meaning of each of the possible parent's syntactic units. This would be helpful for attaching the child unit to the parent units (see Fig. 19.8). These constraints are called selectional constraints, and usually represented in the form of the permissible range of *fillers* in the slots in *frames* that represent the meanings. The potential filler, which is meaning of a child unit, is compared against the constraint by a *fuzzy match* function. A popular way to do this fuzzy match is to compute a weighted distance between the two meanings in a *semantic* or *ontological network* of concepts. The closer the two are in the network, higher is the score assigned to the particular choice.

19.8 Natural Language Parsing

Parsing is carried out to compute the structural description of a sentence. This structural description is assigned by the grammar of the language, and as a precondition it is assumed that sentence is well-formed, i.e., grammatically correct. The process of parsing consists of at least the following activities:

- Mathematical characterization of derivations in a grammar, and those associated with parsing algorithms.

- Computing the time and space complexities of parsing algorithms, with length of input as the length of the input sentence.
- Comparing different grammar formalism and showing equivalences among them whenever possible.
- Combining grammatical and statistical information for improving the efficiency of parser, and ranking the parsers.

The structural descriptions provided by a grammar depend on the formalism to which the grammar belongs. For the well-known context-free grammar (CFG), the structural description is phrase structure grammar.

19.8.1 Parsing with CFGs

A parse-tree describe the structure of a sentence, and is also the record of the sequence of steps of derivation of the sentence. The parse-trees are useful due to following reasons:

- In carrying out semantic analysis, the parsing is an important intermediate stage,
- Grammatically checking the sentence, and
- The parsing is useful in following:
  - Mechanical translation,
  - Question answering, and
  - Information Extraction.

A *syntactic* parser can be thought of as searching through the space of all possible parse-trees to find the correct parse-tree for the given sentence. Before we go through the steps of parsing, let us consider the following rules for grammar.

$S \rightarrow NP\ VP$;a start symbol for sentence can be replaced by noun phrase followed by verb phrase

$S \rightarrow Aux\ NP\ VP$;Aux stands for auxiliary, e.g., do, does

$S \rightarrow VP$

$NP \rightarrow Det\ Nom$;Det is determiners, Nom is for Nomial

$Nom \rightarrow Noun\ Nom$

$Nom \rightarrow N$

$NP \rightarrow proper\text{-}N$;for proper noun

$VP \rightarrow V$

$VP \rightarrow V\ NP$

$VP \rightarrow VP\ PP$;PP is prepositional phrase

$PP \rightarrow Prep\ NP$;Prep is preposition, PP is prepositional phrase

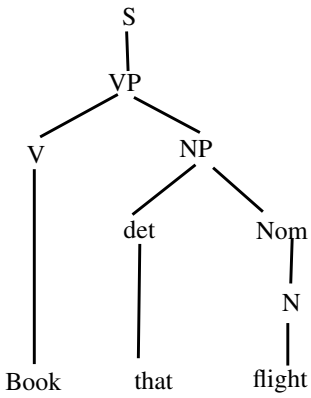

Fig. 19.9 Parsing tree of sentence: "Book that flight"

$$
\begin{aligned}
&Det \rightarrow a \mid an \mid the \\
&N \rightarrow book \mid flight \mid meal \\
&V \rightarrow book \mid include \mid proper \\
&Aux \rightarrow Does \\
&prep \rightarrow from \mid to \mid on \\
&Proper\text{-}N \rightarrow Mumbai \\
&Nom \rightarrow Nom \;\; PP
\end{aligned}
$$

The parse-tree for the sentence "Book that flight" is shown in Fig. 19.9.

Example 19.2 Set of production rules (P) and parsed sentences for different forms of verb-phrase.

$$
\begin{aligned}
S &\rightarrow NP \;\; VP \text{ ; I prefer a morning flight} \\
VP &\rightarrow V \;\; NP \text{ ; prefer a morning flight} \\
VP &\rightarrow V \;\; NP \;\; PP \text{ ; leaves Bombay in the morning} \\
VP &\rightarrow V \;\; PP \text{ ; leaving on Tuesday} \\
PP &\rightarrow prep \;\; NP \text{ ; from New Delhi.}
\end{aligned}
$$

A NP can be location, date, time, or others. Following are other examples of parts of speech (POS).

$$
\begin{aligned}
&N \rightarrow flights \mid breeze \mid trip \mid morning \\
&V \rightarrow is \mid prefer \mid like \mid need \mid want \mid fly \\
&Det \rightarrow a \mid an \mid the \mid this \mid these \mid those
\end{aligned}
$$

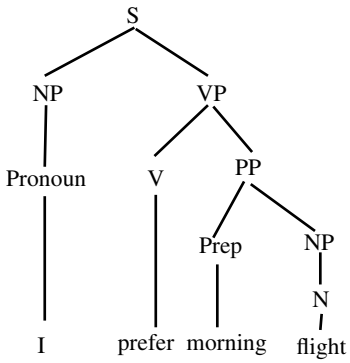

Fig. 19.10 Parse-Tree of sentence: "I prefer morning flight"

$$Pronoun \rightarrow me \mid I \mid you \mid it$$
$$Proper\text{-}N \rightarrow Mumbai \mid Delhi \mid India \mid USA$$
$$Adj \rightarrow cheapest \mid non\text{-}stop \mid first \mid latest \mid other$$
$$Prep \rightarrow from \mid to \mid on \mid near$$
$$Conj \rightarrow and \mid or \mid but$$

The following examples show the substitution rules and with values for each parts-of-speech to be substituted.

$$NP \rightarrow Pronoun(I) \mid proper\text{-}N\ (Mumbai) \mid det\ Nomial$$
(a flight) | $N\ (flight)$
$$VP \rightarrow V\ (do) \mid V\ NP$$(want a flight) | $V\ NP\ PP$
(leaves Delhi in Morning)
$$PP \rightarrow Prep\ NP$$(from Delhi)

Making use of above rules, Fig. 19.10 demonstrates the parsing of sentence "I prefer morning flight."

19.8.2 Sentence-Level Constructions

A sentences can be classified as *declarative*, *imperative*, and *pragmatic*, as follows.

- *Declarative Sentences.* These sentences have structure: $S \rightarrow NP\ \ VP$.
- *Imperative Sentences:* These sentences begin with "VP". For example, "Show the lowest fare," "List all the scores," etc. Following are the production rules for imperative sentences.

$$S \rightarrow VP$$
$$VP \rightarrow V \ \ NP$$

The other substitutions for verb are already discussion in above.

- *Pragmatic Sentences:* The examples of pragmatic sentences are the following:

Can you give me the some information?
Do all these flights have stops?
What flights do you have from Delhi to Mumbai?
What Airlines fly from Delhi?

The pragmatic sentences are governed by substitution rule

$$S \rightarrow Aux \ \ NP \ \ VP.$$

Corresponding to the word at begin of a sentence e.g., "What", the production rule is

$$Wh\text{-}NP \rightarrow What.$$

Hence, for the sentence,

"*What flights do you have from Delhi to Mumbai*?",

can be generated by the production rule,

$$S \rightarrow Wh\text{-}NP\ Aux\ NP\ VP.$$

Many times, the longer sentences are conjuncted together using connectives, e.g., *I will fly to Delhi and Mumbai*. The corresponding rule is

$$VP \rightarrow VP\ and\ VP.$$

19.8.3 Top-Down Parsing

In top-down parsing, the searching is carried out from the root node, substitutions are carried out at every step, and the progressing sentence is compared with the input text sentence to determine whether the sentence generated progressively matches with the original. Figure 19.11 demonstrates the steps of top-down parsing for the sentence "Book that flight".

To carry out the top-down parsing, we expand the tree at each level as shown in the figure. At each level, the trees whose leaves fail to match the input sentence,

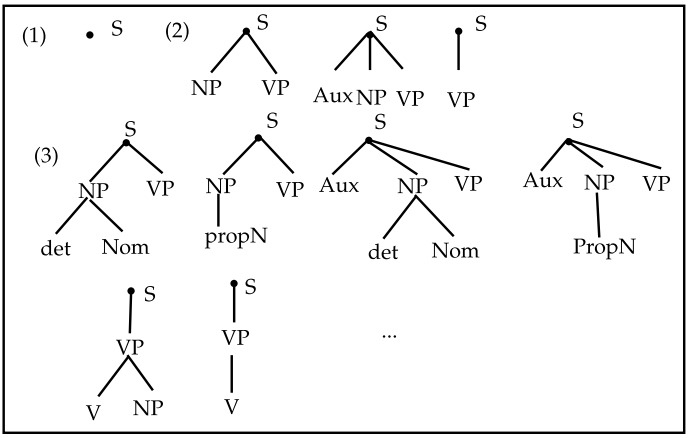

Fig. 19.11 Top-down parsing for the sentence, "Book that flight"

are rejected, leaving behind the trees that represent the successful parses. Going this way, we ultimately construct the original sentence: "Book that flight."

Figure 19.11 shows that, first, root node is created as S, then in step (2), it is expanded to generate three possible sub-trees,

$$S \rightarrow NP\ \ VP,$$
$$S \rightarrow Aux\ \ NP\ \ VP,$$
$$S \rightarrow VP.$$

First of these is expanded at level 2, in step (3), by rule "NP → det Nom", which, we know will not match. Neither, when NP is expanded by NP → PropN in the next sub-tree. So, we need to expand the next choice "S → Aux NP VP" of stage (2). This also does not work. Finally, what works is as follows:

$$S \rightarrow VP$$
$$VP \rightarrow V\ \ NP$$
$$V \rightarrow Book$$
$$NP \rightarrow Det\ \ N$$
$$Det \rightarrow that$$
$$N \rightarrow flight.$$

And, ultimately generates the sentence, "Book that flight."

The bottom-up parsing is other way round, starting from the sentence "Book that flight," and reducing it to start symbol S. Every time we perform a reduction, we reduce the right-hand side of the production $A \rightarrow \alpha$, i.e., α by a variable A.

19.8.4 Probabilistic Parsing

In the discussion above, we drew a sharp line between a correct and an incorrect parses, which was based on whether a terminal node either matched or did not match the next word in the sentence. According to this parsing, a phrase is either acceptable or not. There are situations under which we can relax these requirements, e.g., when we cannot afford to try alternatives exhaustively. The examples are: analysis of connected speech, text segmentation, and identification of words can never be done with complete certainty. At best, one can say that certain sound is more probable than the other. Consequently, one may associate a fractional number with each terminal node, indicating the probability or quality of nodes below that, in respect of forming grammatically correct sentence [5].

In the probabilistic parsing, the non-terminal nodes will be assigned some values, which are sophisticated enough to realize that syntactic and semantic restrictions are taken care of. Hence, a parser that aims to deliver the best analysis even if every analysis violates some constraint, must associate a measure of being grammatical (i.e., acceptable) with the analysis of portions of the sentence. The sentence ultimately associates a measure with the analyses of the complete sentence. As a matter of rule, it is possible to generate an analysis of every sentence with a nonzero acceptability or matching probability, and then select the one having the best analysis.

One simple parser of this type is "best-first" parser, which is based on the modified form of standard *top-down serial* parsing algorithm for context-free grammars. The standard algorithm tries to generate one possible parse-tree until it gets stuck (i.e., on generation a terminal node which does not match the next word in the sentence). In case of stuck, it "backs up" to try another alternative. A *best-procedure*, is like a best-first search that tries all alternatives in parallel. A measure in the numerical value is associated with each alternative path that indicates the likelihood, that this analysis matches the sentence processed so far, and it can be extended to the complete sentence analysis. At each progressive step, the path with the highest likelihood is extended. In the process, if the "measure" of current path falls below that of some other path, the parser shifts its attention to that of other paths.

A CFG (context-free grammar) consists of

terminal words $w^1, w^2, ..., w^V$,
non-terminals $N^1, N^2, ..., N^n$,
start symbol N^1, and
production rules $N^i \rightarrow \alpha^j$,

where α^j is sequence of terminals and non-terminals. Given this, we can define a generative PCFG (probabilistic context-free grammar) as

terminal words $w^1, w^2, ..., w^V$,
non-terminals $N^1, N^2, ..., N^n$,
start symbol N^1, and
production rules $N^i \rightarrow \alpha^j$,

where α^j is sequence of terminals and non-terminals. And can define the rule probabilities as

$$\forall_i \sum_j P(N^i \rightarrow \alpha^j) = 1, \tag{19.3}$$

which shows that for each set of productions having same left-side variable (N^i), the sum of probabilities is unity.

We consider that sentence is represented as sequence of words $w_1 w_2 ... w_m$, and $w_{ab} = w_a ... w_b$ is a subsequence. Let non-terminal N^i dominates subsequence $w_a ... w_b$ is represented by N^i_{ab}, .i.e., N_i is root of sub-tree having children sequence as $w_a ... w_b$. Let $N^i \Rightarrow^* \alpha$. We represent the probability of sentence w_{1n} as

$$P(w_{1n}) = \sum_t P(w_{1n,t}) \tag{19.4}$$

where t is parse-tree of sentence w_{1n}.

Example 19.3 Construct a correct parse-tree for the sentence "I saw an astronomer with telescope," for the following PCFG:

Rule	Probability	Rule	Probability
$S \rightarrow NP\ \ VP$	1.0	$NP \rightarrow NP\ \ PP$	0.20
$PP \rightarrow Prep\ \ NP$	1.0	$NP \rightarrow N$	0.45
$VP \rightarrow V\ \ NP$	0.7	$NP \rightarrow Art\ \ N$	0.35
$VP \rightarrow VP\ \ PP$	0.3	$N \rightarrow telescope$	0.25
$Prep \rightarrow with$	1.0	$N \rightarrow astronomer$	0.15
$Art \rightarrow an$	1.0	$N \rightarrow I$	0.60
$V \rightarrow saw$	1.0		

In this problem, the symbols are as follows:

- Terminals: I, saw, an, astronomer, with, telescope.
- Non-terminals: *S, NP, VP, PP, V, Prep, Art, N*
- Start Symbol: *S*

The sentence above can be generated using two different parse-trees, say T_1 (see Fig. 19.12) and t_2 (see Fig. 19.13).

Probability for parse-tree t_1 is the product of all the probabilities of rules used, which starting from top, and going down through all the levels is given by

$$\begin{aligned} P(t_1) =& 1.0 \times 0.45 \times 0.7 \times 0.60 \times 1.0 \times 0.20 \times 0.35 \times 1.0 \times 1.0 \times 0.45 \times 0.25 \\ =& 0.001488375 \end{aligned}$$

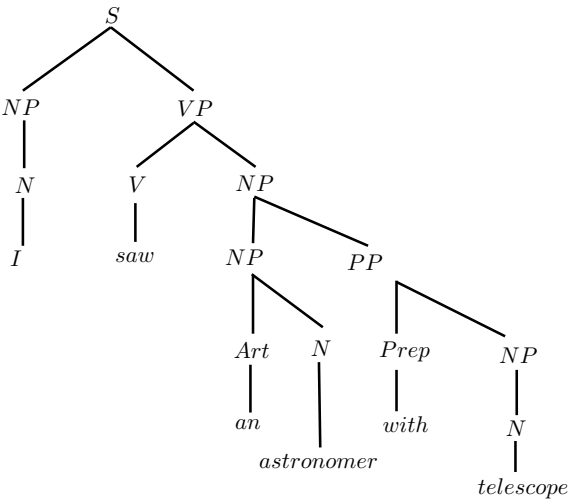

Fig. 19.12 Probabilistic parse-tree t_1 for the sentence "I saw an astronomer with telescope"

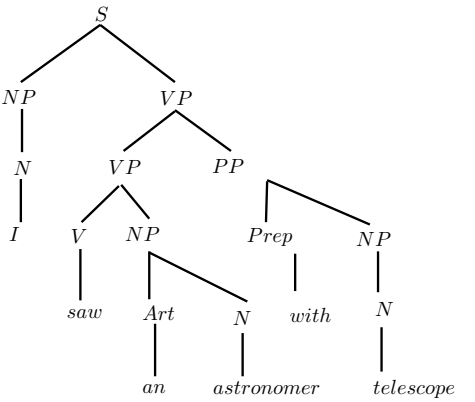

Fig. 19.13 Probabilistic parse-tree t_2 for the sentence "I saw an astronomer with telescope"

Similarly, the probability for parse-tree t_2 is given by

$$\begin{aligned} P(t_2) =& 1.0 \times 0.45 \times 0.3 \times 0.60 \times 1.0 \times 0.35 \times 1.0 \times 0.45 \times 1.0 \times 0.15 \times 0.25 \\ =& 0.00047840625 \end{aligned}$$

Naturally, the parse-tree t_1 is correct, as the probability of its construction is higher. The tree t_1 indicates that the observer (I) is seeing an astronomer carrying a telescope, while t_2 has semantics which indicates that observer (I) is seeing an astronomer with the help of telescope, which obviously is incorrect.

However, the accuracy of the probability of each tree depends on the accuracy of the probabilities of rules used. The probabilities associated with the rules are made

available from the statistics of semantics from a large corpus of natural language text. □

The probabilistic parsing provides a solution to ambiguity in language and grammar, as it is conclusive from the above example. But, this is not necessarily complete, but in partial. The other benefit is that it gives a *probabilistic language model*.

19.9 Information Extraction

We consider the following small text, which we would like to use for *Information Extraction*.

> Firm XYZ is a full service advertising agency specializing in direct and interactive marketing. Located in Bigtown CA, Firm XYZ is looking for an Assistant Account Manager to help manage and coordinate interactive marketing initiatives for a marquee automotive account. Experience in online marketing, automotive and/or the advertising field is a plus. Assistant Account Manager responsibilities ensures smooth implementation of programs and initiatives, helps manage the delivery of projects and key client deliverables …Compensation: 50, 000 − 80, 000, Hiring Organization: Firm XYZ.

Given the above text, the extracted information may be

This information can be loaded into a database and can be queried any number of times to find useful information, at a convenience [4].

The general architecture of an IE (Information Extraction) system is "a cascade of transducers or modules such that, at each step structure is added to the document and, sometimes, it filter relevant information by application of rules" (see Fig. 19.14).

INDUSTRY	Advertising
POSITION	Assistant Account Manager
LOCATION	Bigtown, CA
COMPANY	Firm XYZ
SALARY	50,000–80,000

The majority of current systems follow this general architecture, although specific systems are characterized by their own set of modules. In general, the combination of such modules allow of the functionalities for IE system, discussed below [21].

19.9.1 Document Preprocessing

Preprocessing of the documents can be achieved by a variety of modules such as: text segmenters, filters, tokenizers, lexical analyzers, stemmers, and disambiguators (POS taggers, semantic taggers, etc.)

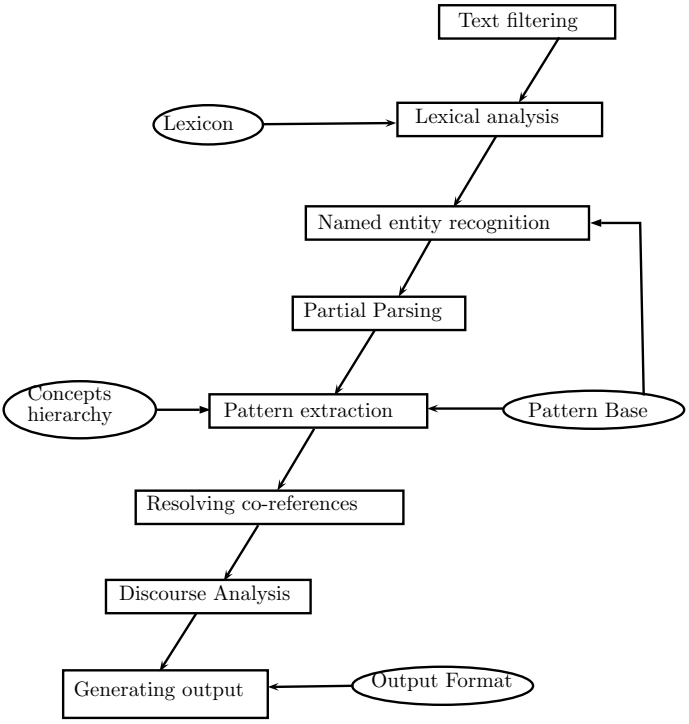

Fig. 19.14 Typical architecture of an IE system

The most relevant document processing activity to IE is Named-Entity recognition, i.e., recognition of proper nouns. The process of named-entity recognition may be performed using finite-state transducers (FST), together with dictionary lookup. These dictionaries are domain-specific, or they are terminological databases.

19.9.2 Syntactic Parsing and Semantic Interpretation

For performing the Information Extraction (IE), a traditional architecture based on Natural Language understanding was used. This method needed full parsing of sentences, and then followed by a semantic interpretation of syntactic structure obtained through parsing. The discourse analysis was carried out as the final step.

A new approach to IE is radically different from traditional, where only the concepts that are within the scope of extraction, are required to be found out in the documents. This leads to simplification of syntactic and semantic analysis due to more restricted, deterministic, and collaborative process. The new approach has the strategy that replaces the traditional parsing, interpretation, and discourse (module)

with a simple *phrasal parser* (parsing based on phrases). The later finds the local phrases. In addition, the discourse module does the job of *event pattern* matching, and merging the of templates. The above approach is useful because, the full parsing is expensive, not robust, hence produces ambiguous results, and cannot manage off-vocabulary conditions.

The current IE systems make use of partial parsing, such that the process of finding the constituent partial structure of a sentence consists one or more cascaded steps onto the text fragments. The generated constituents are tagged as parts-of-speech, like noun/verb, or phrases like prepositional or others. After parsing of the constituents, the system resolves domain-specific dependencies based on the semantic restrictions imposed by the extraction environment. The dependencies are resolved using the following two alternatives.

Pattern Matching

Majority of IE systems follow the approach based on pattern matching, called *named extraction* patterns or IE rules, to resolve the dependencies. In this approach, simplification of the syntax helps in reducing the semantic processing, and that leads ultimately to simplify the pattern matching task. The use of patterns is scenario-specific to recognize both, modifier and argument dependencies between constituents.

Grammatical relations

For representation of the grammatical relations, a graph is constructed (similar to dependency grammars) using the general rules of interpretation, such that previously detected chunks are constructed as nodes, and relations among these nodes are the labeled arcs. Such graphs are useful for reasoning about IE. There are three different types of grammatical relations for IE, as follows.

- First type is defined in the form of general relations of subject, object, and modifier.
- Second type is specialized *modifier relations*, which are specified as temporal relations, and relations concerning to locations.
- Third type is *mediated relations*, i.e., they are mediated by prepositional phrases.

19.9.3 Discourse Analysis

The task of the majority of IE systems is to represent the extracted information from the text in the form of partially filled templates or in some logical forms. Once represented/stored in a specified format, this information can be queried later, like we query a database. Since such information may be incomplete some times, it will result in partially incomplete templates. To recover the missing parts of the information to some extent, merging procedures are used to merge the partial templates, to explore the recovery of missing information.

However, when working with logical forms the IE systems can use traditional semantic interpretation methods in this phase.

19.9.4 Output Template Generation

The output template generation phase is the final stage of IE systems. This stage maps the extracted pieces of information to the required output format. Due to the domain-specific restrictions in the output structure, some inferences can be drawn in this phase. The inference drawing can happen in the following situations:

1. a value from a predefined set is taken as it is by the output slots;
2. a forced instantiation of output slots;
3. when an extracted information generates multiple output templates, it results in inference;
4. some times, the *normalization* of the output slots produces inferences, e. g., when date in the coded form is normalized, it produces, say day or day of month or week, etc. Similarly, when products list is normalized we may get name of items instead of its code, etc.

19.10 NL-Question Answering

The Natural Language Question Answering (QA) derives the common features from NLP (Natural Language Processing) and Information Retrieval (IR). The QA (i.e., NL-QA) promises to deliver "answers", unlike the "hits" by IR. Most of QA is focused questions based on facts like "When did first human land on Moon?", "Who invented the paper clip?", and "How far is the Moon from earth?" These are called "factoid" questions, which can be typically answered using the *named entities* such as people, organizations, measures, dates, locations, etc. [7].

The commonly used approach for answering the factoid questions of open-domain requires the technique of named-entity recognition, together with IR. Typically, major steps of a "traditional" ontology-driven question answering system that is primarily based on IR and named-entity recognition technologies, are shown in Fig. 19.15. Usually, the large ontologies drive the process of QA, which relates the question types to semantic classes of answers. The detailed description of steps for QA is as follows:

1. *Question Analysis.* The first step of QA focus is on the analysis of the question, where knowledge resources are explored to find out the expected semantic type of the answer.
2. Document retrieval. Next, in the *document retrieval* phase, the candidates' documents containing the terms from the questions are retrieved. This retrieval is often done using off-the-shelf IR engines.
3. *Identification of passages.* In the next stage, the system *identifies passages* within these restricted candidate documents that contain a concentration of terms from the question–these regions are likely to contain the answer.
4. *Answer extraction.* Finally, in the *answer extraction* stage, named-entity recognizers identify candidates of the correct semantic type [9].

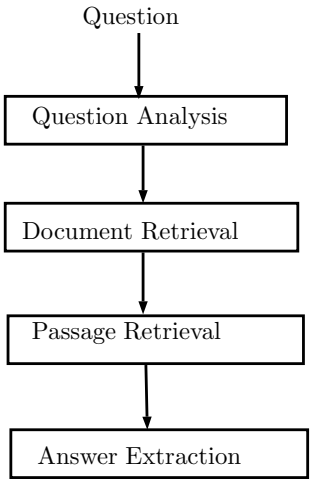

Fig. 19.15 Major steps of a QA system

However, there are challenges of the QA approach discussed above. One is due to complex many-to-many mapping between question types and answer types. As an example, the answer to a "who" question can be name of a person (e.g., "Who invented the light bulb?") or an organization's name (e.g., "Who won the World Series in 2004?"), and there are other possibilities also. On the other side, different question types may map onto the single semantic answer type. For example, the questions – "Where was the world book fair held in 1900?" and "What city hosted the world book fair in 1900?", would map to the same answer. To overcome these challenges of multiple questions with single answer, a system must have elaborate ontological resources which explicitly encode semantic relationships between questions and answers. More advanced techniques such as *abductive inferencing*, *feedback loops*, and *fuzzy matching* of syntactic relations are used, but the factoid question answering is still driven by information retrieval and named-entity recognition technologies, and only on large ontologies.

19.10.1 Data Redundancy Based Approach

The data redundancy based approach depends on statistical regularities, using which it is possible to extract "easy" answers to factoid questions from web. For answering a question, the system makes some connection between the question and the passages containing the answers. Having done this connection at the lexical level, the rest of the job is simple. The meaning of the lexical connection between the question and likely answer carrying passage is the presence of large degree of overlap between these texts. But, in reality, it is not the case very often. This is because, the richness of natural language, and its expressive power provides the facility to create varying

textual forms which have the same meaning (semantics). But, in that case also, there is a good degree of semantics resemblance between the question and answer carrying text. To appreciate this scenario, consider the following question, with two different answers, which though lexically different but have the same semantics.

When did Sikkim become the state of India?

1. Sikkim became a state on 16 May 1975.
2. Sikkim was admitted to India on May 16, 1975.

In the above cases, both the passages contain correct answers, however, it seems obvious that for computer, it would be easier to extract the answer from passage 1 than from passage 2. We note that the task of answering the factoid questions will be far easier for the computer if the answer passage carries the same words as that in the question.

From the point of view of text processing algorithms, answers can be expressed in a variety of different ways. Within text collection, it is possible that the answer to above question lies in passage 2. Though in that text, the answer is not obviously stated. In such cases, since the answers share few words common with the question, a sophisticated natural language processing may be required to find out the relation between them. These processing may typically comprise the following types of domains [7].

Recognizing syntactic alternations,
Collapsing of paraphrases,
Resolving the anaphora,[2]
Making commonsense inferences, and
Performing date calculations.

19.10.2 Structured Descriptive Grammar-Based QA

A special grammar, called SDG (Structured Description Grammar) can be used for information representation and extraction. The basic approach here is data redundancy based. In this, a text sentence (S) is mapped into a transition graph or state diagram, as shown in Fig. 19.16. It shows seven types of structurally different sentences labeled as 1, 2, …, 7, based on the number of *wh*-pronouns and their positions, which can be mapped into this structure. The structure of each sentence explicitly indicates the position of interrogating *wh*-pronouns, *who*, *what*, *where*, *when* and *why*. The positions of these wh-pronouns are invariant [3].

Following examples demonstrate that English-language sentences structures map into the transition diagram of SDG. It can be easily verified that the sentences (a) to (d) are of types 7, 7, 3, 4 respectively.

[2] Anaphora: making use of a pronoun or similar word instead of repeating a word used earlier.

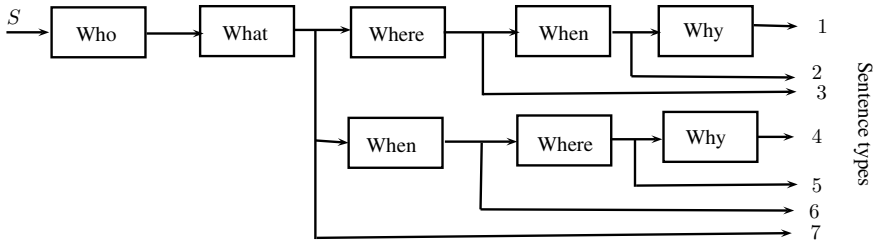

Fig. 19.16 Transition diagram of sentence structure

(a) Akbar | followed a liberal policy for religion.
Who | *What*
(b) Jahangir | Married Nur Jahan.
Who | *What*
(c) Red fort | is located | at Delhi.
Who | *What* | *Where*
(d) Shah Jahan | built Taj Mahal | during his rule | at Agra | in the memory of queen Mumtaj.
Who | *What* | *When* | *Where* | *Why*

It may be noted that the structures of the sentences in SDG are simple English-language sentences. Further, in SDG, i) a question is allowed to carry more than one subquestion, as a sentence has more than one position for wh-pronouns, ii) sub-answers for a question which are distributed in a single document can be aggregated to fill up the slot-like structures in the transition diagram to generate a single answer, and iii) sub-answers for a question which are distributed in multiple passages can also be aggregated to fill up the slot-like structures in the transition diagram to generate a single answer.

19.11 Commonsense-Based Interfaces

To make our computers easier to use, it is required to make them understand the meaning of what we tell them. Usually these efforts are failed because meaning is not one thing but a combination of many. This is because, the activities of human thought engage an enormous collection of different structures and processes [10].

In human understanding, what some thing X means to us depends on how representations of X (in our brain) connects to, that is, having the links to other things we know. If we understand something in only one way then we know it very little, because when something goes wrong, we are left no links to connect that object (sense) with others. But if we use several features, each integrated with its set of related pieces of knowledge, then if one of them fails we can switch to another. This we implement by turning our ideas around in our mind to examine the other pieces

of knowledge from different perspectives, and carry on this process until we find one that works.

When our goal is that our computers must understand us, we will need to equip them with this kind of facility, i.e., they should connect each object/action (sense) to many other senses. For example, the sense "cat" is connected with other senses: color, size, height, no. of legs, tails, eyes, ears, leopard shape, etc. Those ways computers must be equipped with knowledge, like human.

Present Limitations of Computers

For computers to understand like humans, there should be a program with common sense like humans. This is a difficult task, because a typical program has only one way to deal with a problem, so if something goes wrong, the program is totally stuck. However, as humans, we search for alternatives in the event of failure of one approach. The limitation with present-day computers is that they always start from scratch. To make the computers of more worth, we need to supply them a great library of *commonsense knowledge*, which are common even with young children.

The present-day computers are not designed with commonsense in them, nor they have capability to learn from experience. The programs can solve difficult problems in specialized subjects, but even today, computers cannot do the things, which even young children do so easily. The current programs behaves in a limited and inflexible way. Some of the critical differences between the capabilities of computers and human are distinctly explained below.

Vitalist

Computers can do only what they are programmed to do so. The computers have been programmed by people to speak, but in fact the computers do not know what those words mean. The meaning is an intuitive thing, and it cannot be reduced to zeros and ones. The present computers can only do logical things, and meanings are not necessarily logical.

Humanist

The machines are not humanist, as machines have no goals or hopes, nor any fears or phobias. They do not even know that exist, nor they have a sense of accomplishment.

Logician

Since we want that computers should have commonsense, it is more important to understand as what the commonsense is? We need to define it more clearly and precisely. That policy seems alright, but it is wrong when it comes to psychology. Of course, we do not like to be imprecise, but some times, the definitions are not sufficient. So instead, we will take a different approach, and try to design (as opposed to defining) machines that can do the things we want.

19.11.1 Commonsense Thinking

When we say "commonsense thought," we are indicating to things that most people can accomplish, even not knowing many times that they are doing them. Thus, when we hear a sentence like: "Bill told the waiter he wanted some chips," we will infer all possible inferences. Some of them are [10] the following:

- The word "he" means Bill, and not the waiter.
- This event possibly took place in a restaurant, where Bill was a customer, and waiter was close to him at the time. The waiter was working there, waiting for Bill's meal order at that time.
- Bill wants potato chips, not memory or IC chips. No count of chips is mentioned. But, it may be 20–30, and in thousands.
- Bill will start eating the chips soon after he gets them.
- Bill and waiter are both human beings, Bill is old enough to talk, and the waiter is old enough to work.
- This event happened after the date of invention of potato chips (i.e., year 1853).
- It is Bill's assumption that waiter also infers all those the abovementioned things.

Every child learns to use thousands of words at a very young age, but no computer knows even the meaning of some of those, so no computer can even understand a casual conversation. For example, if you mention about "string," any child would know what it means, because it knows that many places are there it can be used, for example, it can be used to tie a cart and pull it. With this, the child would also know things like these:

- An object can be pulled by a string, and cannot be pushed by it.
- It is not good for eating, and you cannot construct a cart using a string.
- Before we put a string into a box, the box needs to be opened.
- And so, on.

Let us find out the size of networks of commonsense knowledge, i.e., network of, say 1000 words, each word having links with other knowledge structures, in different ways. Some links are for objects, while others are for processes. Each such link will, in turn, lead to other links (all in a semantic network), so that whenever some process gets stuck, we usually can find some alternative. Language is not the only feature with such abilities, but each expert skill a person possesses, there must be other similar order of structures, say, for vision, for hearing, for perception, for speech, and for all kinds of physical manipulations, and also for various kinds of social knowledge. So, it may be that we possess millions/billions of units of knowledge.

19.11.2 Components of Commonsense Reasoning

The commonsense thinking we do every day involves a large number of hard-earned ideas, which includes masses of factual knowledge about the problems we are trying

to solve. Not only the learning, but we must also adopt efficient ways to retrieve and apply the relevant knowledge. Many processes gets engaged in the activities of imagining, planing, predicting, and deciding, which also make use of many exceptions and rules. For these, it requires knowledge about how to think, i.e., how to organize and control those processes, also even if the representations are different, it can describe the same situation. In addition, there must be a facility to convert the new experiences into suitable memory structures to represent them. Some of the possible structures are: property lists, frames, frame-arrays, database query languages, explanation-based reasoning, logic programming, rule-based systems, semantic networks, scripts, and stories.

The first step in knowledge representation is to select a representation. Using any specific representation will shortly lead to limitations, which may quickly accumulate, leading the reasoning to ultimately halt. It is usually required to use several different representations for each fragment of the commonsense knowledge about each *idea*, *thing*, or *relationship*. We swiftly change from one method of representation to other methods. And, that depends on which methods are better than other methods for solving problems.

Finally, the following three are the important capabilities we need for characterizing the problems for making use of commonsense reasoning.

Negative Expertise

This deals with ways through which we recognize each of the methods as when they are going to fail. If it becomes possible to recognize the way particular things went wrong, it becomes a clue for deciding what action should be taken next. By knowing the way how each of the methods is likely to fail, it can be used to control the higher activity, e.g., by brain we control the mental activity.

Knowledge Retrieval

Retrieving the relevant information from the commonsense knowledge networks requires appropriate methods to identify, as what problems or situations in the past are having maximum resemblance to the context of present problem. This means, the systems need ways to describe what we are trying to do, and then reason about those descriptions. This is based on the skillful use of analogies.

Self-reflection

Our computers, when implementing commonsense reasoning, are required to keep records which describe *acts* and *thinking* they have recently done, so that they are able to *reflect* on the results of what they tried to do. Also, they must be aware of what they are doing. This is what we call with human as *consciousness*.

How the programs like playing chess or proving theorem are different from commonsense reasoning? Is the reasoning process in a child's brain for fixing blocks of different colors is a game, simpler than that of chess or theorem proving? The answer is No. There is a fundamental difference between these programs, like for chess game or theorem proving versus playing these block games. In fact, the programs of expert systems, like chess or theorem prover, require much less knowledge

skills. However, the children make use of thousands of skills, they manipulate all kinds of things, irrespective of their texture and shapes; they stack the things up and then knock down then, and learn the dynamics of how the stack got scattered. Even for building a small toy house of blocks, one needs to mix and match the knowledge of many kinds: that is, about shapes and colors, speed, space and time, support and balance, stress and strain, and the economics of self-management.

Do computers lack in learning all these things? Not likely. These limitations persist because we have learned to program only in certain ways. Our decades-old approach to solve some problem X has been: find the best way to represent X, find the best way to represent the knowledge needed to solve X, and find the best way to solve X, and so on! These steps lead to write *specialized programs* that cope with solving only that type of problems—called *brittle programs*! This has resulted to millions of specialized programs for solving only that kind of problems, such as playing chess, playing cards, or designing a bridge, banking, etc. In the computer programs, price for best was sacrificed in limiting the resources.

To make the machines deal with commonsense reasoning, we must use multiple ways to represent the knowledge, acquire huge knowledge in that, and find commonsense ways to reason with this. Consider the following example.

> Mary was invited to Jack's party.
> She wondered if he would like a kite.

What leads us to infer that Mary was thinking to take a kite as a gift, but there was no mention of "birthday" or "present" in the first sentence. There should be a suitable representation of phrase "invited to party", which could help to infer that she is thinking of a kite as a suitable gift for Jack.

19.11.3 Representation Structures

It is not well known as which structure of representation is best suitable for any specific purpose. On the contrary, the brain represents a problem in several ways, as well as represents the required knowledge. When one method fails to solve a problem, we quickly switch over to another approach.

Now, to reply to the above question, we do a small analysis. Consider that we are traveling by electric train, and there is a possibility of it being halted in any of the stations due to electricity failure. Assuming that there are 10 railway stations, and it may stuck at any one of them. Therefore, it is not difficult to plan for food and shelter, as there can be a maximum 10 different solutions. So, given a cause of halting of train, there are a maximum of 10 possible effects.

Consider another example, a possibility of meteorite strike on earth of sufficient size, and depending on where it strikes, there are different rescue operations and strategies needed depending on where it strikes. Naturally, for a single cause, there are hundreds or thousands of possible different measures needed(effects), and even more. Naturally, this second problem is more difficult to solve.

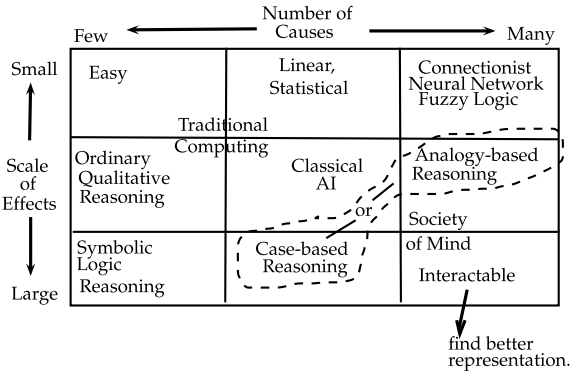

Fig. 19.17 Causal-diversity matrix

The cause-effect matrix (shown in Fig. 19.17) illustrates the relation "$cause \times effect$" to reasoning process, which can be used to arrive at the solution. In the cases, when the causes are only a few, and each cause having a small number of effects as shown in the top-left corner cells of the matrix, such the problems are easy to solve even by exhaustive search methods, or some times there is no need to search, but simply answer is recalled [10].

When the causes are many, each with a small effect, then statistical methods and neural networks may work well, as indicated by the top-right corner cell of the matrix. But, such systems are likely to breakdown if those causes have different characters that interact in hard-to-predict ways.

The symbolic or logical reasoning, shown at the bottom-left corner cell of the matrix, can work well when the causes are few, but corresponding effects are in large number, except that the search may exponential in the case of sequential processes.

It is a rare situation, when many causes correspond to a large number of effects, indicated in the right-bottom of the matrix, such systems are interactable, unless the system is linear. Sometimes it is possible to arrive at solutions by reformulating those difficult problems by switching to different representations, that emphasize fewer and more pertinent features, so that we can work with simpler descriptions.

Except in the right corners of the matrix, there are multiple causes with modest number of effects, the heuristic programs succeed in these conditions, using knowledge combined with controllable search. This is the region of "classical AI" research—the causes and effects are both moderate. Consider the adjacent cells to the bottom and to the right of classical AI. In these regions, analytical methods are usually not helpful, but it may be possible to use use our knowledge and experience to construct and apply analogies, plans, or case-based reasoning methods. Such methods may not work perfectly, but they are frequently useful for practical purposes.

The conclusion of the above discussion is that there cannot be a single type of approach possible for all types of problems. Accordingly, one should not seek a uniform way to represent commonsense knowledge. In fact, there is frequent need to use several representations when we face a difficult problem and then we will need additional knowledge about it. The *causal-diversity* method may help, but eventually

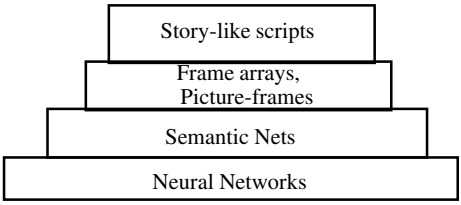

Fig. 19.18 Architecture of representations

it must be replaced by more resourceful, *knowledge based* systems that can generate new representations.

In fact, there is no best way to represent the knowledge. The present limitations of machine intelligence are largely due to our quest for seeking unified theories capable of reasoning well in all situations. The versatility human can emerge from a large-scale architecture of representation, where each of several different representations can help overcome the deficiencies of other representations.

Consequently, the commonsense knowledge in the machines be built, which represents knowledge about so many things like strings, houses, clothing, roads, books; in other words, everything that most children know. For such commonsense knowledge base, we will need ways to link each unit of knowledge to the use, or functions that each unit knowledge can serve. Figure 19.18 shows some typical representation levels, which have been covered in previous chapters.

19.12 Tools for NLP

There are a number of tools that exist, either as open-source or research tools created by some research laboratories, as well as closed source for natural language processing, and speech processing. These tools have built-in functions to perform a number of commonly used tasks, which can be directly called, or using these scripts can be written to perform more complex jobs of NLP and speech processing. For example, for NLP, they can tokenize the given text, can do stemming and POS (parts of speech tagging), can find out word frequencies in given documents, parsing of NL sentences, etc. These inputs can help to compute, for example, $tf \times idf$, which can be helpful in IR (information retrieval), IE (information extraction), text classification, etc. In the following part, we discuss some such tools, which are either open source or they can be obtained on request from respective research laboratories.

19.12.1 NLTK

The NLTK (Natural Language Toolkit) is a collection of Python libraries and programs for *symbolic* and *statistical natural language processing*. It is suited to practitioners as well those who are learning natural language processing (NLP), those who

are conducting research in NLP or areas close to NLP, like, empirical linguistics, cognitive science, AI, IR, and machine learning.

The NLTK has been also used as a teaching tool, as a study tool for individuals, and as a prototyping and building platform for research systems. Python language has been chosen due to its shallow learning curve, its transparent syntax and semantics, and due to its extraordinary capability for handling strings. Python is an interpreted language, which facilitates interactive exploration. It is an object-oriented language, allows data and methods to be encapsulated and reused easily. Python is also available with an extensive standard library, that includes tools for speech processing, natural language processing, numerical processing, and graphical programming [12].

Consider that tasks of stemming and parts-of-speech (POS) tagging are independent, and both operate on sequences of tokens. If the stemming task is done first, the information required for POS tagging is lost. If tagging task is performed first, the stemming process must be able to skip over the tags. If these two tasks are done independent of each other, it becomes difficult to align the resultant texts. Hence, as the combinations of tasks increase, it becomes extremely difficult to manage the data. To address this problem, NLTK version 1.4 onward comes with a new architecture where tokens are based on Python's native dictionary datatype, such that the tokens can have an arbitrary number of *named properties*. The *Tag* and *Stem* are examples of these properties. The NLTK allows for even whole sentence and document to be represented as a single token with *Sub-tokens* attribute that holds sequences of smaller tokens.

A *parse-tree* can also be treated as a token, which has special property/attribute of *Children*. The benefit of this type of architecture in NLTK is that it unifies many different data types, and allows distinct tasks to be run independently. Of course, this architecture comes with an overhead for programmers, because the program needs to keep track of a growing number of property names.

19.12.2 NLTK Examples

Example 19.4 Tokenization of natural language text.

The following commands in Python, with NLP tool NLTK installed, demonstrate the tokenization of a given text into sentence tokens and word tokens. Since there is only one sentence, the sentence token is one only.

```
$ python
Python 2.7.14 (default, Sep 23 2017, 22:06:14)
[GCC 7.2.0] on linux2
Type "help", "copyright", "credits" or "license" for more
                                                    information.
>>> from nltk.tokenize import sent_tokenize, word_tokenize
>>> text="Fundamentals of Artificial Intelligence."
>>> print(sent_tokenize(text))
```

```
    ['Fundamentals of Artificial Intelligence.']

>>> print(word_tokenize(text))
    ['Fundamentals', 'of', 'Artificial', 'Intelligence', '.']
>>>
```

□

Example 19.5 Stemming of given set of words.

Following are the commands for stemming a set of words to reduce them to their stem words. This makes use of the Porter Stemmer algorithm.

```
>>> from nltk.stem import PorterStemmer
>>> ps=PorterStemmer()
>>> words=["python", "pythoning", "pythonize", "pythonly"]
>>> for w in words: print(ps.stem(w))
...
python
python
python
pythonli
>>>
```

□

The parts-of-speech tagging (grammatical tagging) or disambiguation of word-category, is a process of marking-up word in a text (corpus) corresponding to a particular POS. This is carried out based on its definition as well as its context.[3] Various POS in the English language are: noun, verb, adjective, adverb, pronoun, preposition, conjunction, and interjection.

The POS tagging is carried out as part of computational linguistics using some algorithms. These algorithms associate discrete terms, as well as hidden parts of speech, in accordance to a set of descriptive tags. POS-tagging algorithms fall into two distinctive categories: *rule-based* and *stochastic* based. For example, Brill's tagger, one of the first and most widely used English POS taggers, makes use of rule-based algorithms. The following example demonstrates the POS tagging using NLTK.

Example 19.6 Parts-of-speech tagging.

```
>>> import nltk
>>> text=nltk.word_tokenize("Part of speech tagging and POS
                                                    tagger")
```

[3]Context: Relationship with adjacent and related words in a sentence, or phrase, or a paragraph.

```
>>> text
 ['Part', 'of', 'speech', 'tagging', 'and', 'POS', 'tagger']
>>> nltk.pos_tag(text)
 [('part', 'NN'), ('of', 'IN'), ('speech', 'NN'),
 ('tagging', 'NN'), ('and', 'CC'), ('POS', 'NNP'), ('tagger',
                                                    'NN')]
```

□

19.13 Summary

Natural language processing (NLP) is an academic, and technology-based research domain comprising a range of computational techniques for representation and automatic analysis of human languages—a field that is motivated by theory. Automatic analysis of text requires a deep understanding of natural language by machines. However, we are still far away from machines that have this capability. To develop a program that can understand that natural language is a challenge, because most of the natural languages are large and complex, which can have infinitely large number of sentences. In addition, there is ambiguity in natural languages, as many words have more than one meaning, such as *can*, *bear*, *fly*, *orange*, and many others. That gives different meanings to the same sentence, when used in different contexts. Due to this problem, creating programs that understand the NL correctly is a difficult task.

Right from its start, the field of NLP focused on areas like machine translation, IR, IE, question answering, text summarization, topic modeling, and more recently, on opinion mining. Although the *semantic* problems and the actual requirements of NLP were apparent right from the beginning. However, for the more direct applicability of *machine learning techniques*, the strategy adopted was to tackle *syntax* first. This was due to the fact that semantics was considered a more challenging problem.

One of the early representation strategies in NL was FOPL (First-Order Predicate Logic)—a deductive system comprising *axioms* and *rules of inferences*. The FOPL supported to good degree of handling the syntax, but limited order of semantics and *pragmatics*. The syntax is concerned with the well-formedness of the expressions, while semantics specifies what the well-formed expressions mean. The handling of pragmatics is more challenging, they specify how the contextual information can be used to provide a better correlation between different semantics. Properly specifying of pragmatics is essential for the tasks such as WSD (Word-Sens Disambiguation).

Production rule based language model is one of the popular models for the description of Natural language (Chomsky, 1956). Other important models for NLP are Ontology Web Language (OWL), which uses XML-based vocabulary for ontology representation, e.g., definition of classes and relation between them, classes' properties, and constraints on the properties, and constraints on the relations. Network-based models are also common for NLP, like Bayesian networks, which provides joint probability distribution over many interrelated hypotheses, and semantic networks—a

graphical notation for representing knowledge in patterns of interrelated interconnected nodes and arcs.

Some of the common applications of NLP are: Classification of text into categories, Index and search large texts, Automatic translation, Information extraction, Automatic summarization, Question answering, Knowledge acquisition, and Text generations/dialogues.

Syntax of natural language is checked by generating using its grammar, called *phrase structure grammar*. The generating process is a derivation, similar to rigorous proof. Different grammars are classified as per Chomsky hierarchy of grammars, called type 0, 1, 2, and 3, with last one most simple, and first as the most complex.

A sentence of natural language is represented by a syntax-tree, which shows the relations between various constituents of the sentence. Every sentence is supposed to have a corresponding one and only one syntax tree. If a sentence can be generated using more than one syntax-tree, then the language of that sentence as well the grammar are said to be ambiguous—having more than one meaning of the sentence.

Parsing a sentence is the process of computing the structural description of the sentence assigned by the grammar, assuming that the sentence is well-formed. Parsing consists of : Mathematical characterization of derivations in the grammar using a specified algorithm. A parse-tree describe the structure of a sentence, and is also the record of the history of derivation of the sentence. The parse-trees are useful for grammar checking of the sentence, which is an important intermediate stage in semantic analysis, and it plays an important role in applications of language translation, question answering, and information extraction.

Parsing can be top-down—a sentence is generated using start symbol, or it can be bottom-up—a sentence is reduced to start symbol through a sequence of steps. Further, the parsing can be *deterministic*—each production rule has equal weight, in comparison to probabilistic parsing—there is a probability weight associated with each production rule. Usually, in parsing, a sentence is split into NP (noun phrase), and VP (verb phrase). More extensive English grammar is obtained by with the addition of other constituents, such as PP (prepositional phrase), ADJ (adjectives), DET (determiners), ADV (adverb), AUX (auxiliary verb), etc.

Information extraction (IE) systems extract the important information from natural text, and load into databases, that can be queried later a number of times to find useful information, and at convenience.

The general architecture of an IE system is defined as "a cascade of transducers or modules that, at each step, add structure to the documents and, sometimes, filter relevant information, by means of applying rules." Most existing IE systems are based on partial parsing. Generally, the process of finding constituents consists of using a cascade of one or more parsing steps against fragments. The resulting constituents are tagged as noun, verb, prepositional phrases, etc.

Syntactical parsing for IE defines some grammatical relations (subject, object, and modifier), and some specialized modifier relations (temporal, and location). This is made possible using a dependency graph built following general rules of interpretation for the grammatical relations, with previously detected chunks as nodes, and relations. The other jobs performed by IE are: discourse analysis, output template generation, and NL-language question answering.

Question-answering is possible using a special format of grammar called structured descriptive grammar. It is possible to map a sentence with a sequence of *wh-pronouns* (*who, what, when*,...), and consequently, determine the value of a missing wh-pronoun, thus, answering a question.

Preprocessing, is a common phase in NLP applications discussed above. It is carried out on documents using the program modules such as: text zoners, segmenters, filters, tokenizers, lexical analyzers, disambiguators (POS taggers, and semantic taggers), stemmers, etc.

To make our computers to understand us, we need to equip them with adequate knowledge. To help this work, the computer must know what our jobs are. To entertain us they will need to know what their audiences like or need. This requires to create commonsense reasoning in computers. However, it is not well known as which structure of representation is best suitable for any specific purpose. On the contrary, the brain represents a problem in several ways, as well as represents the required knowledge. When one method fails to solve a problem, we quickly switch over to another approach. That means, commonsense reasoning is that we should not seek one uniform way to represent commonsense knowledge. We will frequently need to use several representations when we face a difficult problem and then we will need additional knowledge about it.

Exercises

1. What are the challenges of NLP?
2. Give one example of the following ambiguities:
   a. Phonetic
   b. Syntactic
   c. Pragmatic
3. What are the applications of NLP?
4. Develop the parse-tree to generate the sentence “Rajan slept on the bench” using following rewrite rules:
5. Draw the tree for the following phrases:
   a. after 5 pm.
   b. on Tuesday.
   c. From Delhi.
   d. Any delay at Mumbai.
6. Draw the tree structures for the following sentences:
   a. I would like to fly on Air India.
   b. I need to fly between Delhi and Mumbai.
   c. Please repeat again.

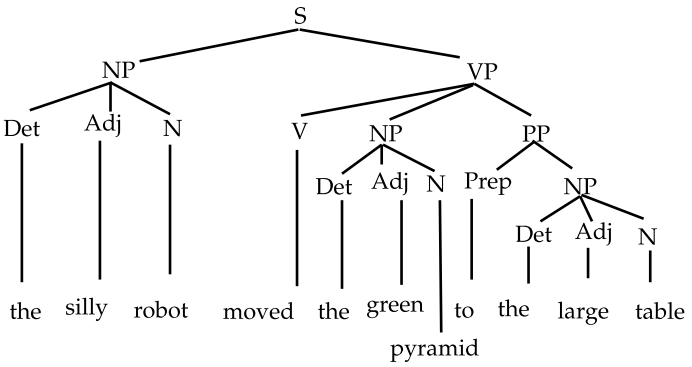

Fig. 19.19 Parse-tree

7. Convert the following passive voice to active voice. Construct the necessary trees. Also write the steps.
   The village was looted by dacoits.

$$
\begin{aligned}
S &\rightarrow NP\ \ VP \\
NP &\rightarrow N \\
NP &\rightarrow Det\ \ N \\
VP &\rightarrow V\ \ PP \\
PP &\rightarrow Prep\ \ NP \\
N &\rightarrow Rajan \mid bench \\
Det &\rightarrow the \\
prep &\rightarrow on
\end{aligned}
$$

8. Given the parse-tree in Fig. 19.19, construct the grammar for this.
9. Construct the grammars and parse-tree for the following sentences.
   a. The boy who was sleeping was awakened.
   b. The boy who was sleeping on the table was awakened.
   c. Jack slept on the table.
10. Construct the parse-trees and resolve the ambiguities in the following sentences using "selectional constraint". Also, specify whether the ambiguities are syntactic, semantic, or some other?
    a. "He saw the man with the horse."
    b. "he saw the man with gun."
    c. "He saw the man with binocular."
11. What are the different types of ambiguities in natural language, like English?

References

1. Cambria E, White B (2014) Jumping NLP curves: a review of natural language processing research. IEEE Computat Intell Mag 5:48–57
2. Chomsky N (1956) Three models for the description of language. IRE Trans Inform Theory 2(3):113–124
3. Chowdhary KR, Bansal VS (2006) Information Extraction from Natural Language Texts. *J of the Institution of Engineers(India)*, 87:14–19
4. Chowdhary KR (2004) Natural Language Processing for Word Sense Disambiguation and Information Extraction. PhD Thesis, JNV University, Jodhpur (India)
5. Jurafsky D, Martin J H (2002) Speech and Language Processing - An Intro to Natural Language Processing, Computational Linguistics, and Speech Recognition. Pearson Education Asia, ISBN 81-7808-594-1
6. Lenat D, Guha R (1989) Building Large Knowledge-Based Systems: Representation and Inference. The Cyc Project. Addison-Wesley, Boston, MA
7. Lin J (2007) An Exploration of the Principles Underlying Redundancy-Based Factoid Question Answering. ACM Trans on Inf Sys 25(2):1–55
8. Miller AM (1995) Wordnet: A Lexical Database for English. Communications of the ACM 38(11):39–41
9. Minsky M (1968) Semantic Information Processing. MA, MIT Press, Cambridge
10. Minsky M (2000) Commonsense-based Interfaces. Communications of the ACM 43(8):67–73
11. Mueller E (1998) Natural Language Processing with Thought-Treasure. Erik T. Mueller, New York
12. https://www.nltk.org/, Cited 19 Dec 2017
13. Pearl J (1985) Bayesian networks: A model of self-activated memory for evidential reasoning. UCLA comput Sci, Irvine, CA: *Tech Rep CSD-850017*
14. Ralph G (1994) Computational Linguistics - An Introduction, Studies in Natural Language Processing, Cambridge Univ Press
15. Reiter R (1980) A logic for default reasoning. Artificial Intelligence 13:81–132
16. Ronald C et al (1997) Survey of the state of art in human language technology – Studies in Natural Language Processing, Cambridge Univ Press
17. Schank R (1975) Conceptual Information Processing. Elsevier Sc Inc, Amsterdam, The Netherlands
18. Singh P (2002) The open mind common sense project. http://www.kurzweilai.net/ Cited 19 Dec 2017
19. Sowa J (1987) Semantic networks. *Encyclopedia of Artificial Intelligence*, S. Shapiro edn, Wiley, New York
20. Baldwin T (2009) et al (2009) Prepositions in Applications: A Survey and Introduction to the Special Issue. *Computational Linguistics*, Vol 35, 2:119–150
21. Turmo J et al (2006) Adaptive Information Extraction. ACM Computing Surveys 38(2):1–47