

6 UNIT 6 INTRODUCTION TO NATURAL LANGUAGE PROCESSING

This unit will cover (A) Overview of Linguistics, Grammars and Languages; (B) Basic Parsing techniques (C) Semantic Analysis & structures and (D) Natural Language generation and systems

An intelligent agent who wants to acquire knowledge needs to be able to understand, at least partially, the natural language that humans speak.

The following are known as information-seeking tasks: text classification, information retrieval and information extraction. They all make use of **language models**: models that predict the probability distribution of language expressions.

The famous Sapir Whorf hypothesis claims that our understanding of the world is strongly influenced by the language we speak. Some languages have several words to express the same concept capable of capturing the concept from slightly different angles. There are also cases in which a certain concept that is well defined in one language, does not 'exist' or cannot be named with one word in other languages. Some linguists also believe that language influences our thoughts or ways of thinking through their grammatical structure or features.

A language can be defined as an infinite set of strings whose formation is governed by a set of rules called grammar. Unlike natural languages, formal languages also have rules that define the meaning or the semantics of the language.

In understanding meaning, probability plays an important role. For example, natural languages are ambiguous. So a sentence may have several meanings, so one considers the probability distribution of a sentence over several meanings.

6.1 An Overview of Linguistics, Grammar and Languages

Linguistics is the academic study of languages whether spoken or written. The academicians that study this are called **linguists**. Linguists study how we learn language (language acquisition), what features characterize a language (phonology, morphology and syntax), how we attach meaning to sounds (semantics), how dictionaries are made (lexicography).

Language can take many forms: spoken, written, but also through signs and non-verbal cues. Speech and writing have several things in common but they are different systems. The most obvious common characteristic is that they both include words. Words in turn have their own structure. Words are made of symbols and these symbols have linguistic values in the sense that they correspond to sounds in spoken language. In English, for example, alphabet letters and speech sounds do not always fit. A symbol 'a' represents at least six sounds: father, ask, all, sensation, around and mare.

Most linguists define language as "a learned system of sounds that have an arbitrary value and meet a social need to communicate". For linguists language is sound not writing. The sounds are called *phones*, groups of sounds are called *morphs*, and they have meaning.

Grammar, in the traditional sense, represents the accumulation of terms, rules, and methods for analysing how language is used. A language may have more than one standard, e.g., the English versus the American standard. Generally the role of a grammar is to devise rules governing the correct usage of a language by legislating constructions that must be adhered to or avoided.

Grammar focuses on the written language.

The main method for syntactically analysing a given written sentence is called **parsing**. Parsing is described in Section 23.2 of Norvig textbook. Parsing is a rote process consisting of several steps:

- 1) Identifying the largest structural components of a sentence: subject and predicate, dependent and independent clauses.
- 2) Classifying each word as one of the eight **parts of speech (POS)**: nouns, pronouns, verbs, adverbs, adjectives, prepositions, conjunctions, interjections.
- 3) Describing individual words in terms of their inflectional or derivational prefixes or suffixes.
- 4) Explaining the relationship of each word to other words in the sentence through a “sentence diagram”.

Context-Free Grammars

Context-free grammars (CFG) are a finite collection of rules which tells us that a certain sentence is grammatically correct (syntactically correct) and what their grammatical structure is.

Here is a CFG written in Prolog which contains 11 context free rules. The ‘- >’ defines a rule. The symbols *s*, *np*, *n*, *det*, *vp*, *v* are called non-terminal symbols. Each of these symbols has a traditional meaning in linguistics: *s* is short for sentence, *np* is short for noun phrase, *vp* is short for verb phrase, and *det* is short for determiner. That is, each of these symbols is shorthand for a grammatical category. The symbols in italics are the terminal symbols or the lexical terms / or words.

```
s -> np vp
np -> n det
vp -> v np
vp -> v
det -> athu
det -> langa
det -> []
n -> abambo
n -> bwino
n -> pano
v -> ali
```

The language generated by a grammar consists of all the strings that the grammar classifies as grammatical. For example the following sentences are grammatically correct with respect to this

CFG:

‘Abambo athu ali bwino’ meaning “Our father is well.”

‘Abambo langa ali pano.’ My father is here.

But also the sentences: “Bwino langa ali pano” which does not make sense in Chichewa.

A context free recogniser is a program which correctly tells us whether or not a string belongs to the language generated by a context free grammar. To put it another way, a recogniser is a program that correctly classifies strings as grammatical or ungrammatical (relative to some grammar).

A context free language is a language that can be generated by a context free grammar. Some languages are context free, and some are not. For example, it seems plausible that English is a context free language. It is probably possible to write a context free grammar that generates all (and only) the sentences that native speakers find acceptable.

TASK 6.1 Do you think Chichewa is a context free language? Explain your answer using as many examples as you want.

Context-Free Grammars in Prolog

Read <http://www.learnprolognow.org/lpnpage.php?pagetype=html&pageid=lpn-htmlse28>

The CFG in the previous section can be written in Prolog using difference lists.

```

s(X,Z):- np(X,Y), vp(Y,Z).
np(X,Z):- n(X,Y),det(Y,Z).
vp(X,Z):- v(X,Y), np(Y,Z).
vp(X,Z):- v(X,Z).
det([athu|W],W).
det([langa|W],W).
det([],_).
n([abambo|W],W).
n([dengu|W],W).
n([bwino|W],W).
n([pano|W],W).
v([ali|W],W).

```

Here's how to recognise sentences:

?- s([abambo,athu,ali,bwino],[]).

True .

You can also generate all sentences that are accepted by this CFG.

?- s(X,[]).

X = [abambo, athu, ali, abambo, athu] ;

X = [abambo, athu, ali, abambo, langa] ;

X = [abambo, athu, ali, abambo] ;

X = [abambo, athu, ali, dengu, athu] ;

X = [abambo, athu, ali, dengu, langa] ;

X = [abambo, athu, ali, dengu]

etc....

To find out if *abambo athu* is a noun phrase we ask:

?- np([abambo, athu],[]).

And we generate all the noun phrases in the grammar as follows:

?- np(X,[]).

TASK 6.2 Write the CFG that accepts the following sentences in Chichewa. Then test your CFG.

“Zinthu zanga zili bwino.”

“Chipinda chake chili uko.”

The way CFG are written using lists is not intuitive. Prolog has a nicer notation to write grammars called the **Definite Clause Grammars**.

```
s --> np, vp.  
np --> n,det.  
vp --> v, np.  
vp --> v.  
det --> [athu].  
det --> [langa].  
det --> [].  
n --> [abambo].  
n --> [dengu].  
n --> [bwino].  
n --> [pano].  
v --> [ali].
```

You can test this grammar with the same queries as before: `?- s([abambo,athu,ali,bwino],[]).` The two grammars are equivalent as Prolog translates the DFG into CFG. To see this run the following query.

```
?- listing(s).  
s(A, B) :-  
    np(A, C),  
    vp(C, B).
```

TASK 6.3 Rewrite your CFG from the previous task using now DFG instead of CFG.

TASK 6.4 (Exercise 7.3 from online tutorial learnprolognow.org) Let $a^n b^{2n}$ be the formal language which contains all strings of the following form: an unbroken block of a s of length n followed by an unbroken block of b s of length 2n , and nothing else. For example, abb , aabbbb , and aaabbbbb belong to $a^n b^{2n}$, and so does the empty string. Write a DCG that generates this language.

Chomsky Normal Forms (CNF)

A CFG is said to be in Chomsky Normal form if the production rules are all of the following forms:

- $A \rightarrow a$ (a terminal symbol)
- $A \rightarrow BC$ (at most two symbols on the right hand side RHS)
- $S \rightarrow \epsilon$ (null string)

where A,B,S are non-terminal symbols and a is a terminal symbol and ϵ is the empty symbol.

The conversion to Chomsky Normal Form has four main steps:

1. Get rid of all ϵ productions, where ϵ is the empty symbol. For example, for step 1: if $P \rightarrow Ax B$ with both A and B nullable, add productions $P \rightarrow xB|Ax|x$. After this, delete all productions with empty RHS.
2. Get rid of all productions where RHS is one variable. A unit production is where RHS has only one symbol. Consider the production $A \rightarrow B$. Then for every production $B \rightarrow \alpha$, add the production $A \rightarrow \alpha$.
3. Replace every production that is too long by shorter productions. For example, if have production $A \rightarrow BCD$, then replace it with $A \rightarrow BE$ and $E \rightarrow CD$.
4. Move all terminals to productions where RHS is one terminal. For every terminal on the right of a non-unit production, add a substitute variable. For example, replace production $A \rightarrow bC$ with productions $A \rightarrow BC$ and $B \rightarrow b$.

Example 6.1 (Source <https://www.cs.bgu.ac.il/~auto191/wiki.files/9a.pdf>):

Consider the CFG:

$$S \rightarrow aXbX$$

$$X \rightarrow aY \mid bY \mid \epsilon$$

$$Y \rightarrow X \mid c$$

After the elimination of ϵ we obtain

$$S \rightarrow aXbX \mid abX \mid aXb \mid ab$$

$$X \rightarrow aY \mid bY \mid a \mid b \quad (\text{Note that the variable } X \text{ is nullable; and so therefore is } Y).$$

$$Y \rightarrow X \mid c$$

After elimination of the unit production $Y \rightarrow X$, we obtain:

$$S \rightarrow aXbX \mid abX \mid aXb \mid ab$$

$$X \rightarrow aY \mid bY \mid a \mid b$$

$$Y \rightarrow aY \mid bY \mid a \mid b \mid c$$

Now, break up the RHSs of S ; and replace a by A , b by B and c by C wherever not units:

$$S \rightarrow EF \mid AF \mid EB \mid AB$$

$$X \rightarrow AY \mid BY \mid a \mid b$$

$$Y \rightarrow AY \mid BY \mid a \mid b \mid c$$

$$E \rightarrow AX$$

$$F \rightarrow BX$$

$$A \rightarrow a$$

$$B \rightarrow b$$

$$C \rightarrow c$$

TASK 6.5 (Convert the following CFG into Chomsky Normal Form:

$$S \rightarrow AbA$$

$$A \rightarrow Aa \mid \epsilon$$

TASK 6.6 (Is it true that any context-free grammar can be automatically transformed into Chomsky Normal Form? Explain your answer.)

6.2 Parsing Techniques: The CYK Algorithm

The word problem of a language L asks for deciding whether a given input word w is a member of L . The Cocke–Younger–Kasami-Algorithm (CYK or CKY) is a highly efficient **parsing algorithm** for context-free grammars written in CNF.

The algorithm uses a “dynamic programming” or “table filling algorithm”.

How the CYK-allgorithm works

Fix a context-free grammar $G(N, T, P, S)$ in Chomsky normal form.

The algorithm decides for an input w whether $w \in L(G)$.

For a given input $w = a_1 \cdots a_n \in T^n$, let for all $1 \leq i \leq j \leq n$

$$w_{ij} = a_i a_{i+1} \cdots a_j.$$

The algorithm computes inductively for $j - i = 0, \dots, n - 1$ the sets

$$N_{ij} = \{X \in N : X \xrightarrow{G,*} w_{ij}\}.$$

Obviously, the word w is in $L(G)$ if and only if S is in N_{1n} .

(Source: <https://www.math.uni-heidelberg.de/logic/ss17/formsprach/fs20-cky-algorithm-41.pdf>)

The CYK algorithm starts by constructing a triangular table so that each row corresponds to one length of sub-strings. The following are then calculated:

- $X_{i,i}$ is the set of variables A such that $A \rightarrow w_i$ is a production of G
- Compare at most n pairs of previously computed sets:

$$(X_{i,i}, X_{i+1,j}), (X_{i,i+1}, X_{i+2,j}) \dots (X_{i,j-1}, X_{j,j})$$

$X_{1,5}$				
$X_{1,4}$	$X_{2,5}$			
$X_{1,3}$	$X_{2,4}$	$X_{3,5}$		
$X_{1,2}$	$X_{2,3}$	$X_{3,4}$	$X_{4,5}$	
$X_{1,1}$	$X_{2,2}$	$X_{3,3}$	$X_{4,4}$	$X_{5,5}$
w_1	w_2	w_3	w_4	w_5

Table for string 'w' that has length 5

Example 6.2¹:

Show the CYK Algorithm with the following example:

– CNF grammar **G**

- $S \rightarrow AB \mid BC$
- $A \rightarrow BA \mid a$
- $B \rightarrow CC \mid b$
- $C \rightarrow AB \mid a$

– **w** is baaba

– Question Is **baaba** in $L(G)$?

Step 1: Construct the triangular table

Step 2: Calculate the bottom row

{B}	{A, C}	{A, C}	{B}	{A, C}
b	a	a	b	a

$S \rightarrow AB \mid BC$
 $A \rightarrow BA \mid a$
 $B \rightarrow CC \mid b$
 $C \rightarrow AB \mid a$

Step 3: Calculate second bottom row

$X_{1,2} = (X_{i,i}, X_{i+1,j}) = (X_{1,1}, X_{2,2}) = \{B\}\{A,C\} = \{BA, BC\}$ Look for production rules to generate BA or BC. $X_{1,2} = \{S, A\}$

$X_{2,3} = (X_{i,i}, X_{i+1,j}) = (X_{2,2}, X_{3,3}) = \{A,C\}\{A,C\} = \{AA, AC, CA, CC\} = \{B\}$

$X_{3,4} = (X_{i,i}, X_{i+1,j}) = (X_{3,3}, X_{4,4}) = \{A,C\}\{B\} = \{AB, CB\} = \{S, C\}$

$X_{4,5} = (X_{i,i}, X_{i+1,j}) = (X_{4,4}, X_{5,5}) = \{B\}\{A,C\} = \{BA, BC\} = \{S, A\}$

Step 4: Calculate the third bottom row

$X_{1,3} = (X_{i,i}, X_{i+1,j})(X_{i+1,i+1}, X_{i+2,j}) = (X_{1,1}, X_{2,3})(X_{1,2}, X_{3,3}) = \{B\}\{B\} \cup \{S, A\}\{A, C\} = \{BB, SA, SC, AA, AC\} = \{\emptyset\}$ (The empty set)

¹ Source <https://web.cs.ucdavis.edu/~rogaway/classes/120/winter12/CYK.pdf>

$$X_{2,4} = (X_{i,i}, X_{i+1,j})(X_{i,i+1}, X_{i+2,j}) = (X_{2,2}, X_{3,4})(X_{2,3}, X_{4,4}) = \{A, C\} \{S, C\} \cup \{B\} \{B\} = \{AS, AC, CS, CC, BB\} = \{B\}$$

$$X_{3,5} = (X_{i,i}, X_{i+1,j})(X_{i,i+1}, X_{i+2,j}) = (X_{3,3}, X_{4,5})(X_{3,4}, X_{5,5}) = \{A, C\} \{S, A\} \cup \{S, C\} \{A, C\} = \{AS, AA, CS, CA, SA, SC, CA, CC\} = \{B\}$$

Step 5/6: Calculate the last two bottom rows. The final table is:

$\{S, A, C\}$	$\leftarrow X_{1,5}$			
\emptyset	$\{S, A, C\}$			
\emptyset	$\{B\}$	$\{B\}$		
$\{S, A\}$	$\{B\}$	$\{S, C\}$	$\{S, A\}$	
$\{B\}$	$\{A, C\}$	$\{A, C\}$	$\{B\}$	$\{A, C\}$
	b	a	a	b
				a

A visual simulation of the CYK algorithm can be found here: <https://www.xarg.org/tools/cyk-algorithm/>

TASK 6.7 Follow the example below and check that the solution shown is the correct one.

Example Application of the CYK-algorithm

Consider the language $L = \{ww^R : w \in \{0,1\}^+\}$ of all palindromes over the binary alphabet of even nonzero length.

$j =$	1	2	3	4	5	6	$i =$
$a_1 = 0$	$\{X_0\}$	$\{S\}$	$\{R_1\}$	\emptyset	\emptyset	$\{S\}$	1
$a_2 = 0$	$\{X_0\}$	\emptyset	\emptyset	$\{S\}$	$\{R_0\}$	\emptyset	2
$a_3 = 1$	$\{X_1\}$	$\{S\}$	$\{R_0\}$	\emptyset	\emptyset	\emptyset	3
$a_4 = 1$	$\{X_1\}$	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	4
$a_5 = 0$	$\{X_0\}$	$\{S\}$	\emptyset	\emptyset	\emptyset	\emptyset	5
$a_6 = 0$	$\{X_0\}$	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	6

Let G be a grammar in Chomsky normal form that generates L and has the rules

$$S \rightarrow X_0 R_0 \mid X_1 R_1 \mid X_0 X_0 \mid X_1 X_1,$$

$$R_0 \rightarrow S X_0, \quad R_1 \rightarrow S X_1,$$

$$X_0 \rightarrow 0, \quad X_1 \rightarrow 1,$$

Applying the CKY-algorithm to G and the input $w = 001100$ yields the sets N_{ij} as shown in the table on the left.

THEOREM (CYK): The CYK Algorithm correctly computes X_{ij} for all i and j ; thus w is in $L(G)$ if and only if S is in X_{1n} . The running time of the algorithm is $O(n^3)$.

TASK 6.8 Consider the following CNF grammar G with 4 production rules. Is $w=ababa$ in $L(G)$?
 $S \rightarrow AB \mid BC$

$$A \rightarrow BA \mid a$$

$$B \rightarrow CC \mid b$$

$$C \rightarrow AB \mid a$$

The CYK Parser is the most efficient one but requires the grammar to be in CNF. There are other parsers as summarised in the table below².

ALGORITHM	MAIN FEATURES	USE CASES
CYK	<ul style="list-style-type: none"> great worst-case performance ($O(n^3)$) grammars require special CNF form 	Specific problems (e.g., membership problem)
Earley	<ul style="list-style-type: none"> great worst-case performance ($O(n^3)$), usually linear performance can handle all kinds of grammars and languages 	Parser generators that must be able to handle all grammars and languages
LL	<ul style="list-style-type: none"> easy to implement less powerful than most other algorithms (e.g., cannot handle left-recursive rules) historically was the most popular choice 	Hand built parsers; parser generators that are easier to build
LR	<ul style="list-style-type: none"> hard to implement (some variants are easier than others) powerful (can handle most grammars, some variants can handle all of them) great performance (usually linear, more powerful variants can have worst-case performance of $O(n^3)$) 	Parser generators with the best performance
Packrat (PEG)	<ul style="list-style-type: none"> linear time of performance uses a special format designed for parsing computer languages 	Simple, but powerful parsers or parser generators for computer languages
Parser Combinator	<ul style="list-style-type: none"> modular and easy to build comparatively bad performance ($O(n^4)$ in the worst case) 	Parser generators that are easy to use and understand
Pratt	<ul style="list-style-type: none"> peculiar and lesser known algorithm no need for a grammar 	Parsing expressions or other cases in which precedence is not dictated by the order of appearance

TASK 6.9 Do some research and explain how the LL parser works. Give 2 examples that show how this algorithm runs. Compare it with the CYK algorithm. What are the similarities and differences?

² Source <https://tomassetti.me/guide-parsing-algorithms-terminology/>