Great Place To Work. Certified MAR 2021–FEB 2022 INDIA

UPES

UNIVERSITY OF TOMORROW

nirf RANKING 2021
Ranked amongst **top 100** universities in **India**

Accredited **Grade** 'A' by NAAC

QS STARS RATING SYSTEM 2020
QS 5 Star Rating for Academic Development, Employability, Facilities and Program Strength

Perfect score of **150/150** as a testament to exceptional E-Learning methods

University of the Year (North India) awarded by ASSOCHAM

SAFE GUARD
Certified for **safety and hygiene** by Bureau Veritas

Lexical analysis

- Identifying and analysing the structure of words.
- Lexicon: Collection of words and phrases in a language.
- Lexical analysis: Divide the whole chunk of txt into paragraphs, sentences, and words.
- Lexical analysis is the process of trying to understand what words mean, intuit their context, and note the relationship of one word to others.

Token name	Sample token values
identifier	x, color, UP
keyword	**if**, **while**, **return**
separator	}, (, ;
operator	+, <, =
literal	true, 6.02e23, "music"
comment	*/* Retrieves user data */, // must be negative*

Syntactic parsing

- Syntactic analysis is the process of analyzing natural language with the rules of formal grammar. We applied grammatical rules only to categories and groups of words, not applies to individual words.
- The syntactic analysis basically assigns a semantic structure to text. It is also known as syntax analysis or parsing.
- Syntactic analysis is the third phase of NLP. The purpose of this phase is to draw exact meaning, or dictionary meaning from the text. Syntax analysis checks the text for meaningfulness comparing to the rules of formal grammar. For example, the sentence like “hot ice-cream” would be rejected by semantic analyzer.

Syntactic parsing

- we will understand the techniques used to analyze the syntax or the grammatical structure of sentences.
- Lexical analysis is aimed only at data cleaning and feature extraction using techniques like stemming, lemmatization, correcting misspelled words, etc. But, in syntactic analysis, we target the roles played by words in a sentence, interpreting the relationship between words and the grammatical structure of sentences.
- For example, let’s take these two sentences :
- Delhi is the capital of India.
- Is Delhi the of India capital.

Syntactic parsing

- Both sentences have the same words, but only the first one is syntactically correct and understandable. Basic lexical processing techniques cannot make this distinction. Therefore, more sophisticated syntax processing techniques are needed to understand the relationship between individual words in a sentence.
- It involves analysis of words in the sentence for grammar and arranging words in a manner that shows the relationship among the words.
- For example, School go a boy (grammatical structure is not correct.), "hot ice cream"
- Syntactic analysis or parsing may be defined as the process of analyzing the strings of symbols in natural language conforming to the rules of formal grammar.

Difference between Lexical and Syntactic analysis

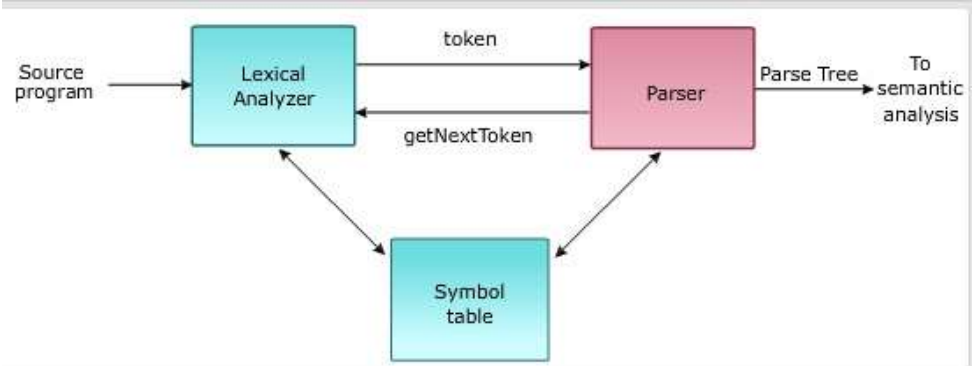
- The aim of lexical analysis is in Data Cleaning and Feature Extraction with the help of techniques such as
  - Stemming,
  - Lemmatization,
  - Correcting misspelled words, etc.
- In syntactic analysis, our target is to :
  - Find the roles played by words in a sentence,
  - Interpret the relationship between words,
  - Interpret the grammatical structure of sentences.

Difference between Lexical and Syntactic analysis

- The syntactical analysis looks at the following aspects in the sentence which lexical doesn't :
- **Words order and meaning:** Syntactical analysis aims to find how words are dependent on each other. Changing word order will make it difficult to comprehend the sentence.
- **Retaining stop-words:** Removing stop-words can altogether change the meaning of a sentence.
- **Morphology of words:** Stemming, lemmatization will bring the words to its base form, thus modifying the grammar of the sentence.
- **Parts-of-speech of words in a sentence:** Identifying correct part-of-speech of a word is important. Example — 'cuts on his face' (Here 'cuts' is a noun) & 'he cuts an apple' (Here, 'cuts' is a verb).

What is a Parser?

- One of the most important parts of syntactic processing is parsing. It means to break down a given sentence into its ‘grammatical constituents’.
- It is defined as the software component that is designed for taking input text data and gives a structural representation of the input after verifying for correct syntax with the help of formal grammar. It also generates a data structure generally in the form of a parse tree or abstract syntax tree or other hierarchical structure.



Interaction Between Lexical Analyzer and Parser

What is a Parser?

- We can understand the relevance of parsing in NLP with the help of the following points:
- The parser can be used to **report any syntax error**.
- It helps to recover from commonly occurring errors so that the processing of the remainder of the program can be continued.
- A **parse tree is created** with the help of a parser.
- The parser is used **to create a symbol table**, which plays an important role in NLP.
- A parser is also used to produce intermediate representations (IR).

What is a Parser?

- Suppose you ask a question to a question-answering (QA) system, such as Siri or Alexa, the following question: “Who won the Formula 1 championship in 2019?”
- The QA system can meaningfully respond only if it can understand that the phrase ‘Formula 1 championship’ relates to the phrase ‘in 2019’. The phrase ‘in 2019’ refers to a specific time frame, and thus significantly revises the question. Finding such dependencies or relationships between the phrases of a sentence can be achieved through parsing techniques.
- “The quick brown fox jumps over the table”. The sentence is divided into three main constituents:
- ‘The quick brown fox’ is a **noun phrase**
- ‘jumps’ is a **verb phrase**
- ‘over the table’ is a **prepositional phrase**.

Different levels of syntactic analysis

Part-of-speech (POS) tagging: A word can be tagged as a noun, verb, adjective, adverb, preposition, etc. depending on its role in the sentence. Assigning correct tags such as nouns, verbs, adjectives, etc. is one of the most fundamental functions in syntactic analysis.

For example, you ask Alexa or google home a question — "Ok Google, where can I get a permit to travel between different states?". Now, the word 'permit' may possibly have two POS tags — a noun and a verb. In the phrase 'I need a work permit', the correct tag of 'permit' is 'noun'. On the other hand, in the phrase "Please permit me to go outside.", The word 'permit' is a 'verb'.

Assigning the correct POS tag helps us to better understand the intended meaning of a phrase or sentence and is thus an important part of syntactic processing.

"*Although POS tagging helps us in identifying the linguistic role of the word in a sentence, it wouldn't enable us to understand how these words are related to each other in a sentence.*"

Different levels of syntactic analysis

- **Constituency parsing:** To deal with the complexity and ambiguity of natural language, we first need to identify and define commonly seen grammatical patterns. The first step in understanding grammar is to divide words into groups, called constituents, based on their grammatical role in the sentence.

"Consider a sentence 'Ishan — read — an article on Syntactic Analysis'.

The most common constituencies in English are **Noun Phrases (NP)**, **Verb Phrases (VP)**, and **Prepositional Phrases (PP)**.

- **Dependency Parsing:** In dependency grammar, constituencies (such as NP, VP, etc.) do not form the main elements of grammar, but dependencies are established between words themselves.
- Dependency parsing is the process of analyzing the grammatical structure of a sentence based on the dependencies between the words in a sentence.

Different levels of syntactic analysis

- Let's take an example sentence '**man saw dog**". The dependencies can be said as follows: '**man**' is the subject of the sentence (the one who is doing something); '**saw**' is the main verb (something that is being done); while '**dogs**' is the object of '**saw**' (to whom something is being done).
- So, the basic idea of dependency parsing is based on the fact that each sentence is about something, and usually involves a subject (the doer), a verb (what is being done) and an object (to whom something is being done).
- In general, Subject-Verb-Object (SVO) is the basic word order in current English. Of course, many sentences are more complex to fall into this simple SVO structure, although sophisticated dependency parsing techniques are able to handle most of them.

Different types of Parsers

- A parser is a procedural interpretation of grammar. It tries to find an optimal tree for a particular sentence after searching through the space of a variety of trees.
  - Recursive Descent Parser
  - Shift-reduce Parser
  - Chart Parser
  - Regexp Parser
- **Recursive Descent Parser:** It is one of the most straightforward forms of parsing.

  Some important points about recursive descent parser are as follows:

Different types of Parsers

- It follows a top-down process.
- It tries to check whether the syntax of the input stream is correct or not.
- It scans the input text from left to right.
- The necessary operation for these types of parsers is to scan characters from the input stream and match them with the terminals with the help of grammar.

"In this kind of parsing, the parser starts constructing the parse tree from the start symbol and then tries to transform the start symbol to the input. The most common form of top-down parsing uses recursive procedure to process the input. The main disadvantage of recursive descent parsing is backtracking."

Different types of Parsers

- **Shift-reduce Parse:** Some important points about shift-reduce parser are as follows:
- It follows a simple bottom-up process.
- It aims to find the words and phrases sequence that corresponds to the right-hand side of a grammar production and replaces them with the left-hand side of the production.
- In simple words, this parser starts with the input symbol and aims to constructs the parser tree up to the start symbol.

"In this kind of parsing, the parser starts with the input symbol and tries to construct the parser tree up to the start symbol."

Different types of Parsers

- **Chart Parse:** Mainly, this parser is useful for ambiguous grammars, including grammars of natural languages.
- It applies the concept of dynamic programming to the parsing problems.
- **Regexp Parser:** It uses regular expressions to parse the input sentences and produce a parse tree out of this.

What is Derivation?

- We need a sequence of production rules in order to get the input string. The derivation is a set of production rules. During parsing, we have to decide the non-terminal, which is to be replaced along with deciding the production rule with the help of which the non-terminal will be replaced.
- **Types of Derivation**

The two types of derivations, which can be used to decide which non-terminal to be replaced with the production rule:

- **Left-most Derivation**

In the left-most derivation, the sentential form of input is scanned and replaced from the left to the right. In this case, the sentential form is known as the left-sentential form.

- **Right-most Derivation**

In the left-most derivation, the sentential form of input is scanned and replaced from right to left. In this case, the sentential form is called the right-sentential form.

Concept of Grammar

- Grammar is very essential and important to describe the syntactic structure of well-formed programs. They denote syntactical rules for conversation in natural languages. Linguistics have attempted to define grammars since the inception of natural languages like English, Hindi, etc.

A mathematical model of grammar was given by Noam Chomsky in 1956, which is effective for writing computer languages.

Mathematically, a grammar G can be formally written as a 4-tuple (N, T, S, P) where −

N or V_N = set of non-terminal symbols, i.e., variables.

T or $\sum$ = set of terminal symbols.

S = Start symbol where $S \in N$

P denotes the Production rules for Terminals as well as Non-terminals. It has the form $\alpha \rightarrow \beta$, where α and β are strings on $V_N \cup \sum$ and least one symbol of α belongs to V_N

Phrase Structure or Constituency Grammar

- Phrase structure grammar, introduced by Noam Chomsky, is based on the constituency relation. That is why it is also called constituency grammar.

Fed raises interest rates.

Example

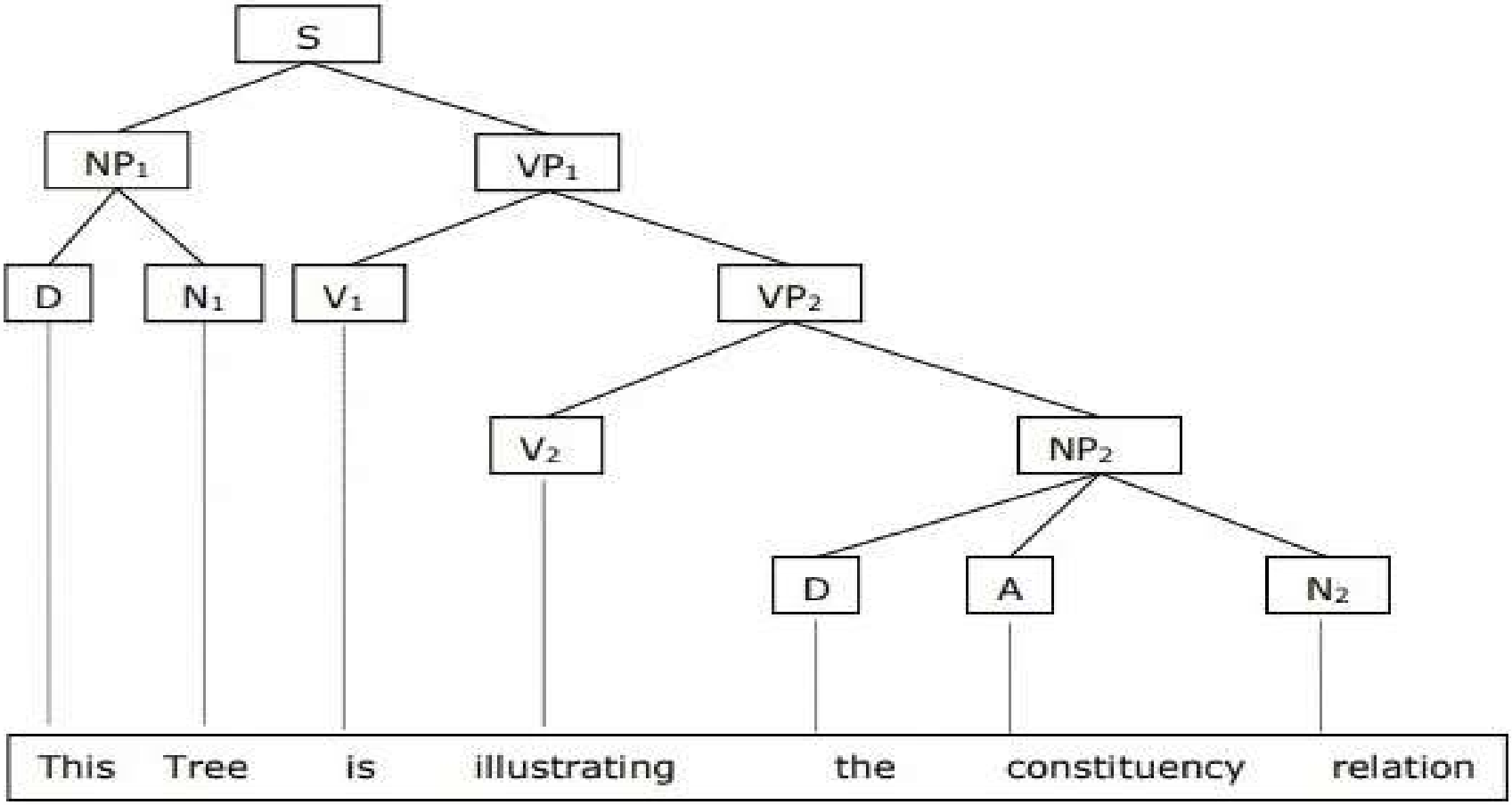
- All the related frameworks view the sentence structure in terms of constituency relation.
- Phrase structure organizes words into nested constituents.
- The constituency relation is derived from the subject-predicate division of Latin as well as Greek grammar.
- The basic clause structure is understood in terms of noun phrase NP and verb phrase VP.
- **Distribution:** a constituent behaves as a unit that can appear in different places
    - John talked [to the children] [about drugs].
    - John talked [about drugs] [to the children].
    - *John talked drugs to the children about.

Substitution expansion pro-forms:

- I sat [on the box / right on top of the box /there].

We can write the sentence **"This tree is illustrating the constituency relation"** as follows

Phrase Structure or Constituency Grammar



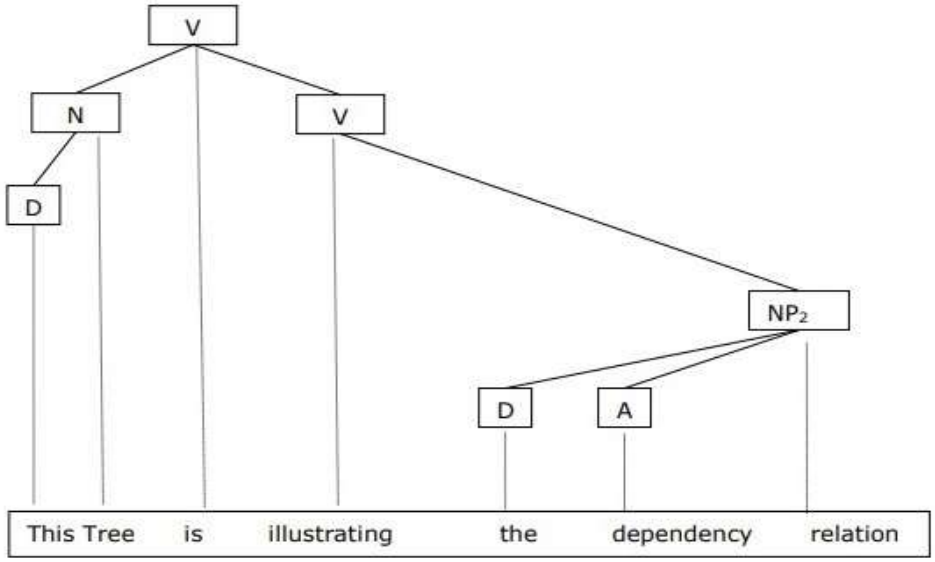
Dependency Grammar

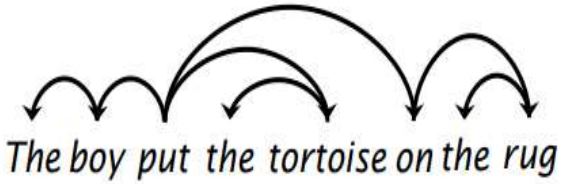
- Dependency grammar (DG) is opposite to the constituency grammar because it lacks phrasal nodes.

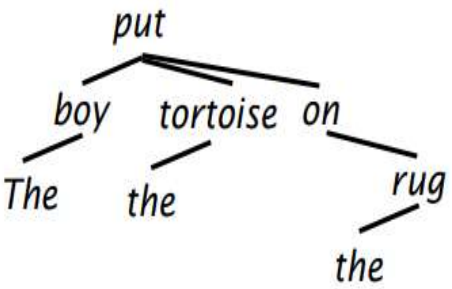
Example

- In DG, the linguistic units, i.e., words are connected to each other by directed links.
- The basic clause structure is understood in terms of noun phrase NP and verb phrase VP.

We can write the sentence **"This tree is illustrating the constituency relation"** as follows





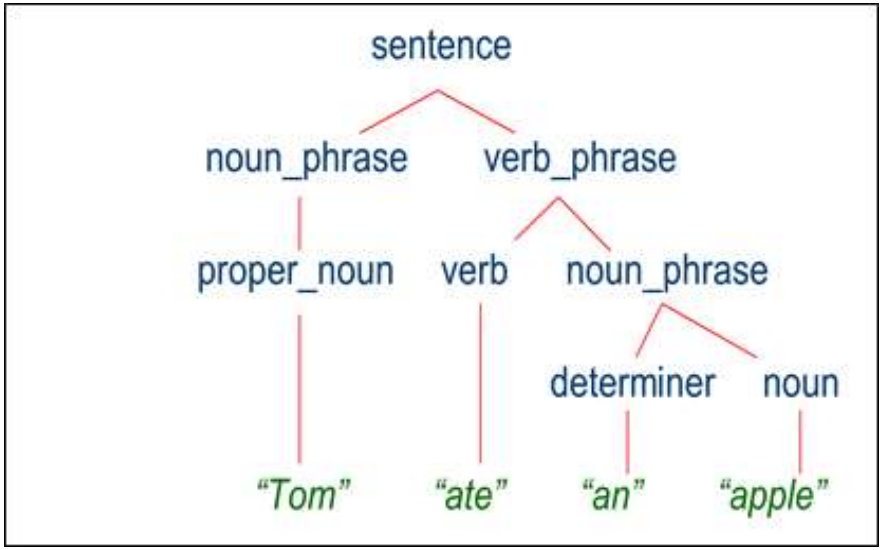


Parsing

- Parsing in NLP is the process of determining the syntactic structure of a text by analyzing its constituent words based on an underlying grammar (of the language).

See this example grammar below, where each line indicates a rule of the grammar to be applied to an example sentence "Tom ate an apple".

```
sentence -> noun_phrase, verb_phrase
noun_phrase -> proper_noun
noun_phrase -> determiner, noun
verb_phrase -> verb, noun_phrase
proper_noun -> [Tom]
noun -> [apple]
verb -> [ate]
determiner -> [an]
```



Existing parsing approaches are basically statistical, probabilistic, and machine learning-based. Some notable tools to use for parsing are: Stanford parser (The Stanford Natural Language Processing Group), OpenNLP (Apache OpenNLP Developer Documentation) etc.

Probabilistic Context Free Grammars:

A phrase structure grammar

S → NP VP
VP → V NP
VP → V NP PP
NP → NP NP
NP → NP PP
NP → N
NP → *e*
PP → P NP

N → people
N → fish
N → tanks
N → rods
V → people
V → fish
V → tanks
P → with

people fish tanks
people fish with rods

Probabilistic Context Free Grammars:

Phrase structure grammars = context free grammars (CFGs)

- G = (T, N, S, R)
  - T is a set of terminal symbols
  - N is a set of nonterminal symbols
  - S is the start symbol ($S \in N$)
  - R is a set of rules/productions of the form $X \rightarrow \gamma$
    - $X \in N$ and $\gamma \in (N \cup T)^*$

- A grammar G generates a language L.

Probabilistic Context Free Grammars:

Phrase structure grammars in NLP

- G = (T, C, N, S, L, R)
  - T is a set of terminal symbols
  - C is a set of preterminal symbols
  - N is a set of nonterminal symbols
  - S is the start symbol ($S \in N$)
  - L is the lexicon, a set of items of the form $X \rightarrow x$
    - $X \in P$ and $x \in T$
  - R is the grammar, a set of items of the form $X \rightarrow \gamma$
    - $X \in N$ and $\gamma \in (N \cup C)^*$
- By usual convention, S is the start symbol, but in statistical NLP, we usually have an extra node at the top (ROOT, TOP)
- We usually write *e* for an empty sequence, rather than nothing

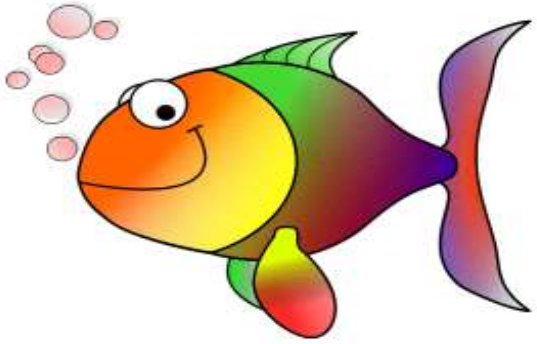
Probabilistic or stochastic context free grammars (PCFGs)

- G = (T, N, S, R, P)
  - T is a set of terminal symbols
  - N is a set of nonterminal symbols
  - S is the start symbol (S ∈ N)
  - R is a set of rules/productions of the form $X \rightarrow \gamma$
  - P is a probability function
    - P: R → [0,1]
    - $\forall X \in N, \sum_{X \rightarrow \gamma \in R} P(X \rightarrow \gamma) = 1$
- A grammar G generates a language model L.

$$\sum_{\gamma \in T^*} P(\gamma) = 1$$

Probabilistic or stochastic context free grammars (PCFGs)

S → NP VP	1.0	N → *people*	0.5
VP → V NP	0.6	N → *fish*	0.2
VP → V NP PP	0.4	N → *tanks*	0.2
NP → NP NP	0.1	N → *rods*	0.1
NP → NP PP	0.2	V → *people*	0.1
NP → N	0.7	V → *fish*	0.6
PP → P NP	1.0	V → *tanks*	0.3
		P → *with*	1.0



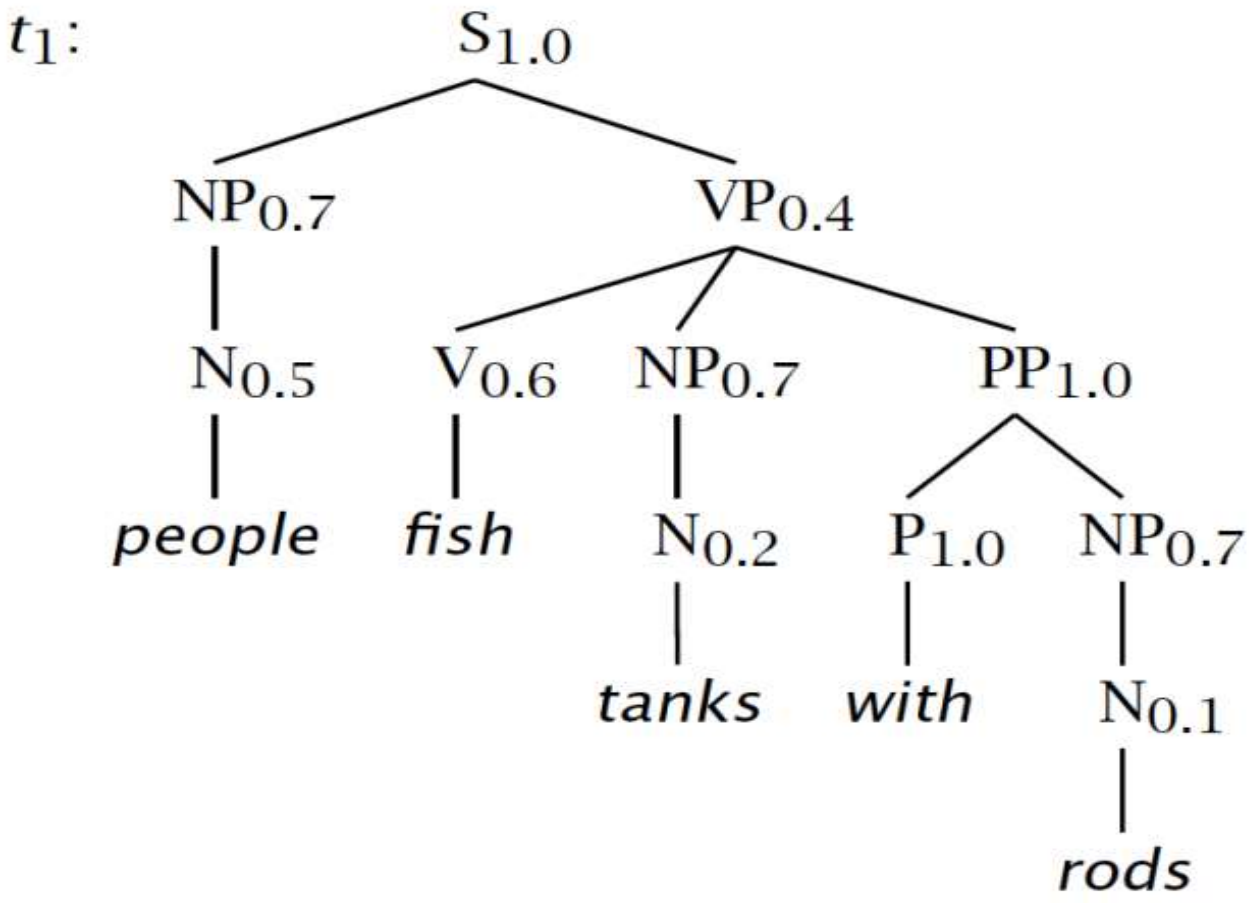
[With empty NP removed
so less ambiguous]

The probability of trees and strings

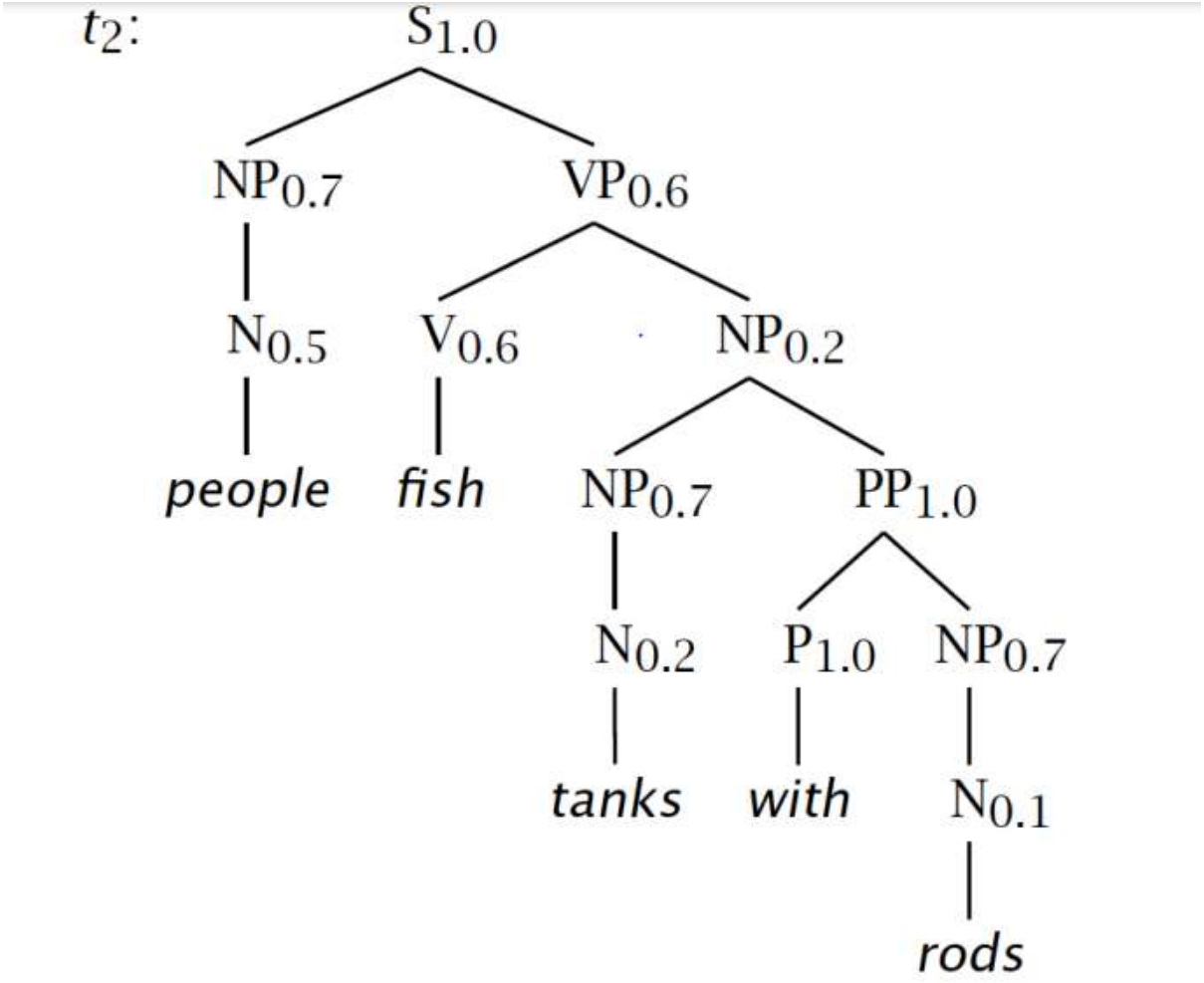
- P(t) – The probability of a tree t is the product of the probabilities of the rules used to generate it.
- P(s) – The probability of the string s is the sum of the probabilities of the trees which have that string as their yield

$$P(s) = \Sigma_j P(s, t) \text{ where } t \text{ is a parse of } s$$
$$= \Sigma_j P(t)$$

The probability of trees



The probability of trees



Tree and String Probabilities

- s = *people fish tanks with rods*
- $P(t_1) = 1.0 \times 0.7 \times 0.4 \times 0.5 \times 0.6 \times 0.7 \times 1.0 \times 0.2 \times 1.0 \times 0.7 \times 0.1$
  $= 0.0008232$

  Verb attach
- $P(t_2) = 1.0 \times 0.7 \times 0.6 \times 0.5 \times 0.6 \times 0.2 \times 0.7 \times 1.0 \times 0.2 \times 1.0 \times 0.7 \times 0.1$
  $= 0.00024696$

  Noun attach
- $P(s) = P(t_1) + P(t_2)$
  $= 0.0008232 + 0.00024696$
  $= 0.00107016$

Grammar Transforms

Chomsky Normal Form

- All rules are of the form $X \rightarrow Y\ Z$ or $X \rightarrow w$
  - $X, Y, Z \in N$ and $w \in T$
- A transformation to this form doesn't change the weak generative capacity of a CFG
  - That is, it recognizes the same language
    - But maybe with different trees
- Empties and unaries are removed recursively
- n-ary rules are divided by introducing new nonterminals ($n > 2$)

A phrase structure grammar

S → NP VP

VP → V NP

VP → V NP PP

NP → NP NP

NP → NP PP

NP → N

NP → *e*

PP → P NP

N → *people*

N → *fish*

N → *tanks*

N → *rods*

V → *people*

V → *fish*

V → *tanks*

P → *with*

Chomsky Normal Form steps

S → NP VP
S → VP
VP → V NP
VP → V
VP → V NP PP
VP → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

N → *people*
N → *fish*
N → *tanks*
N → *rods*
V → *people*
V → *fish*
V → *tanks*
P → *with*

Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V
S → V
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

N → *people*
N → *fish*
N → *tanks*
N → *rods*
V → *people*
V → *fish*
V → *tanks*
P → *with*

Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

N → *people*
N → *fish*
N → *tanks*
N → *rods*
V → *people*
S → *people*
V → *fish*
S → *fish*
V → *tanks*
S → *tanks*
P → *with*

Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

N → *people*
N → *fish*
N → *tanks*
N → *rods*
V → *people*
S → *people*
VP → *people*
V → *fish*
S → *fish*
VP → *fish*
V → *tanks*
S → *tanks*
VP → *tanks*
P → *with*

Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP PP
NP → P NP
PP → P NP

NP → *people*
NP → *fish*
NP → *tanks*
NP → *rods*
V → *people*
S → *people*
VP → *people*
V → *fish*
S → *fish*
VP → *fish*
V → *tanks*
S → *tanks*
VP → *tanks*
P → *with*
PP → *with*

Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V @VP_V
@VP_V → NP PP
S → V @S_V
@S_V → NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP PP
NP → P NP
PP → P NP

NP → *people*
NP → *fish*
NP → *tanks*
NP → *rods*
V → *people*
S → *people*
VP → *people*
V → *fish*
S → *fish*
VP → *fish*
V → *tanks*
S → *tanks*
VP → *tanks*
P → *with*
PP → *with*

A phrase structure grammar

S → NP VP

VP → V NP

VP → V NP PP

NP → NP NP

NP → NP PP

NP → N

NP → *e*

PP → P NP

N → *people*

N → *fish*

N → *tanks*

N → *rods*

V → *people*

V → *fish*

V → *tanks*

P → *with*

Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V @VP_V
@VP_V → NP PP
S → V @S_V
@S_V → NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP PP
NP → P NP
PP → P NP

NP → *people*
NP → *fish*
NP → *tanks*
NP → *rods*
V → *people*
S → *people*
VP → *people*
V → *fish*
S → *fish*
VP → *fish*
V → *tanks*
S → *tanks*
VP → *tanks*
P → *with*
PP → *with*

Chomsky Normal Form

- You should think of this as a transformation for efficient parsing
- With some extra book-keeping in symbol names, you can even reconstruct the same trees with a detransform
- In practice full Chomsky Normal Form is a pain
  - Reconstructing n-aries is easy
  - Reconstructing unaries/empties is trickier

Four people need to cross a rickety bridge at night. Unfortunately, they have only one torch and the bridge is too dangerous to cross without one. The bridge is only strong enough to support two people at a time. Not all people take the same time to cross the bridge. Times for each person: 1 min, 2 mins, 7 mins and 10 mins. What is the shortest time needed for all four of them to cross the bridge?

Thank You