Unit 8
Context Free Grammar for English

- Geoff Pullum noted in a talk that “almost everything most educated Americans believe about English grammar is wrong”.

- In this chapter we make a preliminary stab at addressing some of these gaps in our knowledge of grammar and syntax, as well as introducing some of the formal mechanisms that are available for capturing this knowledge

- The word **syntax** comes from the Greek *s´yntaxis*, meaning SYNTAX “setting out together or arrangement”,
  - refers to the way words are arranged together.

- Various syntactic notions are seen till now:
  - The regular languages - offered a simple way to represent the ordering of strings of words
  - Compute probabilities - for these word sequences.
  - Part-of-speech categories - could act as a kind of equivalence class for words

- The present one introduces sophisticated notions of syntax and grammar that go well beyond these simpler notions.

- Three main new ideas are introduced in this chapter:
  - **Constituency**
  - **Grammatical relations**
  - **Subcategorization and dependency**

Constituency

- The fundamental idea of constituency is that groups of words may behave as a single unit or phrase, called a **constituent.**

  - For example we will see that a group of words called a **noun phrase** often acts as a unit;
  - noun phrases include single words like ***she*** or ***Michael*** and phrases like ***the house***, ***Russian Hill***, and *a* ***well-weathered, three-story structure***.

- How do words group together in English? Consider the **noun phrase**, a sequence of words surrounding at least one noun.
  - Examples of noun phrases:
    - three parties from Brooklyn
    - Harry the Horse
    - a high-class spot such as Mindy’s
    - the Broadway coppers
    - they

- How do we know that these words group together (or “form constituents”)? **One piece of evidence is that they can all appear in similar syntactic environments**, ***for example before a verb***.
    - three parties from Brooklyn *arrive*. . .
    - a high-class spot such as Mindy’s *attracts*. . .
    - the Broadway coppers *love*. . .
    - they *sit*

- But while the whole noun phrase can occur before a verb, this is not true of each of the individual words that make up a noun phrase.
- Thus to correctly describe facts about the ordering of these words in English, we must be able to say things like “**Noun Phrases can occur before verbs**”.

- Other kinds of evidence for constituency come from what are called **preposed** or **postposed** constructions.
- For example, the prepositional phrase ***on September seventeenth*** can be placed in a number of different locations in the following examples, including **preposed at the beginning, and postposed at the end:**

On September seventeenth, I’d like to fly from Atlanta to Denver
I’d like to fly ***on September seventeenth*** from Atlanta to Denver
I’d like to fly from Atlanta to Denver ***on September seventeenth***

- But again, while the entire phrase can be placed differently, the individual words making up the phrase cannot be:

  * **On September**, I’d like to fly **seventeenth** from Atlanta to Denver
  * **On** I’d like to fly **September seventeenth** from Atlanta to Denver
  * I’d like to fly **on September** from Atlanta to Denver **seventeenth**

- This chapter will introduce the use of **context-free grammars**, a formalism that will allow us to model these constituency facts.

Grammatical relations

- Grammatical relations are a formalization of ideas from traditional grammar such as SUBJECTS and OBJECTS, and other related notions.

- In the following sentence the noun phrase ***She*** is the SUBJECT and ***a mammoth breakfast*** is the OBJECT:

  - **She ate a mammoth breakfast**

Subcategorization and **dependency relations**

- **Subcategorization** and **dependency relations** refer to certain kinds of relations between words and phrases.

- For example the verb ***want*** can be followed by an infinitive, as in ***I want to fly to Detroit***, or a noun phrase, as in ***I want a flight to Detroit***.

- But the verb ***find*** cannot be followed by an infinitive **(**I found to fly to Dallas*).** These are called facts about the ***subcategorization*** of the verb

- None of the syntactic mechanisms that we’ve discussed up until now can easily capture such phenomena.

- They can be modeled much more naturally by grammars that are based on **context-free grammars (CFG)**

- CFG is integral to many computational applications including grammar checking, semantic interpretation, dialogue understanding, and machine translation.

- They are powerful enough to express sophisticated relations among the words in a sentence and efficient algorithms exist for parsing sentences

Context-Free Grammar

- The most commonly used mathematical system for modeling constituent structure in English and other natural languages is the **Context-Free Grammar**, or **CFG**.
- Context free grammars are also called **Phrase-Structure Grammars**, and the formalism is equivalent to what is also called **Backus-Naur Form** or **BNF**.

- A context-free grammar consists of :
  - a **set of rules** or productions, each of which expresses the ways that symbols of the language can be grouped and ordered together, and
  - a **lexicon** of words and symbols. For example, the following productions express that a **NP (or noun phrase),** can be composed of either a **Proper Noun or a determiner (Det)** followed by a **Nominal;** a Nominal can be one or more Nouns.

    - $NP \rightarrow Det\ Nominal$
    - $NP \rightarrow ProperNoun$
    - $Nominal \rightarrow Noun \mid Nominal\ Noun$

- Context-free rules can be **hierarchically embedded**, so we can combine the previous rules with others like the following which express facts about the lexicon:

  - *Det → a*
  - *Det → the*
  - *Noun → flight*

- The symbols that are used in a CFG are divided into two classes.
  - The symbols that correspond to words in the language (“the”, “nightclub”) are called ***terminal symbols***;
    - the lexicon is the s et of rules that introduce these terminal symbols
  - The symbols that express clusters or generalizations of these are called ***non-terminals***

- In each context free rule, the item to the right of the ***arrow (→)*** is an ordered list of **one or more terminals and non-terminals**, while to the left of the arrow is a **single non-terminal symbol** expressing some cluster or generalization.

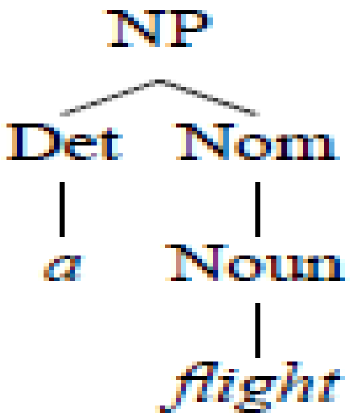
- Notice that in the lexicon, the non-terminal associated with each word is its lexical category, or part-of-speech

- A CFG can be thought of in two ways:
  - as a device for generating sentences, and
  - as a device for assigning a structure to a given sentence.

- As a generator, we can read the **→arrow** as “rewrite the symbol on the left with the string of symbols on the right

- The string ***a flight*** can be derived from the non-terminal NP
- This sequence of rule expansions is called a **derivation of the string of words**
- It is common to represent a derivation by a ***parse tree*** (commonly shown inverted with the root at the top)

- In the parse tree for ***a flight*** , we say that the node *NP* **immediately dominates** the node ***Det*** and the node ***Nom***.
- We say that the node *NP* **dominates** all the nodes in the tree (*Det*, *Nom*, *Noun*, *a*, *flight*).
- The formal language defined by a CFG is the set of strings that are derivable from the designated **start symbol**.



- Each grammar must have one designated start symbol which is often called ***S***.
- Since context-free grammars are often used to define sentences, ***S*** is usually interpreted as the “**sentence**” node, and the set of strings that are derivable from ***S*** is the set of sentences in some simplified version of English.

- A sentence can consist of a noun phrase followed by a **verb phrase**
- A verb phrase in English consists of a verb followed by assorted other things; for example, one kind of verb phrase consists of a verb followed by a noun phrase
- Or the verb phrase may have a verb followed by a noun phrase and a prepositional phrase
- Or the verb may be followed by a prepositional phrase alone
- A prepositional phrase generally has a preposition followed by a noun phrase

S → NP VP	I prefer a morning flight
VP → Verb NP	prefer a morning flight
VP → Verb NP PP	leave Boston in the morning
VP → Verb PP	leaving on Thursday
PP → Preposition NP	from Los Angeles (locations, times, dates)

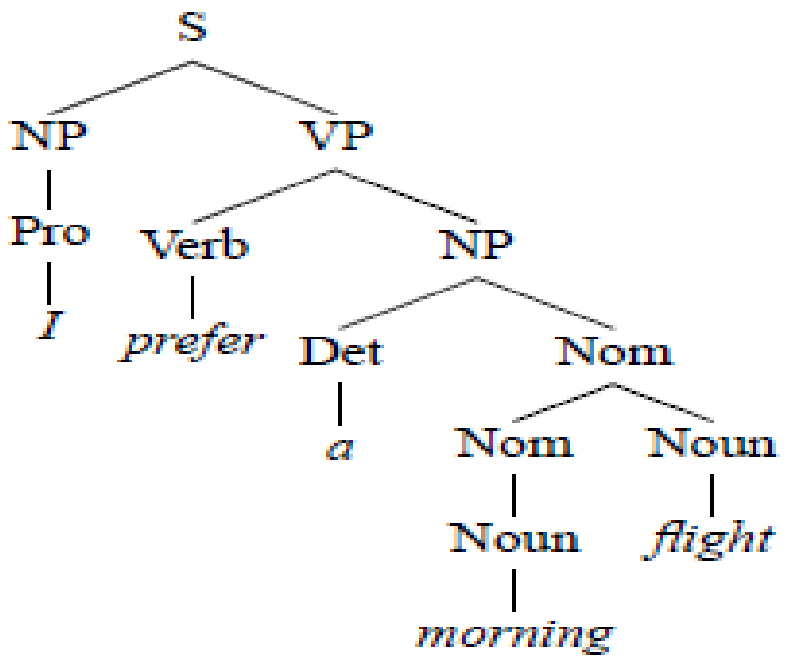
Context-Free Grammer Example

$$
\begin{aligned}
\mathit{Noun} &\rightarrow \mathit{flights} \mid \mathit{breeze} \mid \mathit{trip} \mid \mathit{morning} \mid \ldots \\
\mathit{Verb} &\rightarrow \mathit{is} \mid \mathit{prefer} \mid \mathit{like} \mid \mathit{need} \mid \mathit{want} \mid \mathit{fly} \\
\mathit{Adjective} &\rightarrow \mathit{cheapest} \mid \mathit{non-stop} \mid \mathit{first} \mid \mathit{latest} \\
&\quad \mid \mathit{other} \mid \mathit{direct} \mid \ldots \\
\mathit{Pronoun} &\rightarrow \mathit{me} \mid \mathit{I} \mid \mathit{you} \mid \mathit{it} \mid \ldots \\
\mathit{Proper\text{-}Noun} &\rightarrow \mathit{Alaska} \mid \mathit{Baltimore} \mid \mathit{Los\ Angeles} \\
&\quad \mid \mathit{Chicago} \mid \mathit{United} \mid \mathit{American} \mid \ldots \\
\mathit{Determiner} &\rightarrow \mathit{the} \mid \mathit{a} \mid \mathit{an} \mid \mathit{this} \mid \mathit{these} \mid \mathit{that} \mid \ldots \\
\mathit{Preposition} &\rightarrow \mathit{from} \mid \mathit{to} \mid \mathit{on} \mid \mathit{near} \mid \ldots \\
\mathit{Conjunction} &\rightarrow \mathit{and} \mid \mathit{or} \mid \mathit{but} \mid \ldots
\end{aligned}
$$

Ten examples from the ATIS corpus (L_0)

S	→	*NP VP*	I + want a morning flight
NP	→	*Pronoun*	I
	|	*Proper-Noun*	Los Angeles
	|	*Det Nominal*	a + flight
Nominal	→	*Nominal Noun*	morning + flight
	|	*Noun*	flights
VP	→	*Verb*	do
	|	*Verb NP*	want + a flight
	|	*Verb NP PP*	leave + Boston + in the morning
	|	*Verb PP*	leaving + on Thursday
PP	→	*Preposition NP*	from + Los Angeles

The grammar for L_0, with example phrases for each rule.



The parse tree for "I prefer a morning flight" according to grammar L_0.

- It is sometimes convenient to represent a parse tree in a more compact format called bracketed notation, essentially the same as LISP tree representations

- Example: The bracketed representation of the parse tree (previous slide)

[S [NP [Pro I]] [VP [V prefer] [NP [Det a] [Nom [N morning] [Nom [N flight]]]]]]

- A CFG like that of $\mathbf{L_0}$ defines a formal language.

- A formal language is a set of strings

- Sentences (strings of words) that can be derived by a grammar are in the formal language defined by that grammar, and are called ***grammatical sentences***
- Sentences that cannot be derived by a given formal grammar are not in the language defined by that grammar, are referred to as ***ungrammatical***

- In linguistics, the use of formal languages to model natural languages is called ***generative grammar,*** since the language is defined by the set of possible sentences "**generated**" by the grammar.

Formal definition of context-free grammar

- A context-free grammar G is defined by four parameters N, ∑, P, S (technically "is a 4-tuple"):
  - ***N*** a set of **non-terminal symbols (or variables)**
  - **∑** a set of **terminal symbols (disjoint from *N*)**
  - **R** a set of **rules** or productions, each of the form $A \rightarrow \beta$, where A is a nonterminal, β is a string of symbols from the infinite set of strings $(\Sigma \cup N)*$
  - **S** is a designated **start symbol**

Formal definition of context-free grammar

Convention:

Capital letters like A, B, and S	**Non-Terminals**
S	**Start Symbol**
Lower-case Greek letters like α, β and γ	**Strings drawn from $(\Sigma \cup N)*$**
Lower-case Roman letters like u, v, and w	**Strings of terminals**

- A language is defined via the concept of **derivation**
- One string derives another one if it can be rewritten as the second one via some series of **rule applications**

- If $A \rightarrow \beta$ is a production of P and α and γ are any strings in the set $(\Sigma \cup N)*$, then we say that **$\alpha A\gamma$ directly derives $\alpha\beta\gamma$, or $\alpha A\gamma \Rightarrow \alpha\beta\gamma$**

- Derivation is then a generalization of direct derivation:

  Let $\alpha 1, \alpha 2, \ldots, \alpha m$ be strings in $(\Sigma \cup N)*, m \geq 1$, such that $\alpha 1 \Rightarrow \alpha 2, \alpha 2 \Rightarrow \alpha 3, \ldots, \alpha m-1 \Rightarrow \alpha m$

  We say that **$\alpha 1$ derives αm, or $\alpha 1 * \Rightarrow \alpha m$.**

- The language L_G generated by a grammar ***G*** can be defined as the set of strings composed of terminal symbols which can be derived from the designated start symbol S.

  $L_G = \{w | w$ is in $\Sigma*$ and $S * \Rightarrow w\}$

- The problem of mapping from a string of words to its parse tree is called **parsing**

SOME GRAMMAR RULES FOR ENGLISH

Sentence-Level Constructions

- There are a large number of constructions for English sentences, but four are particularly common and important:
- ***Declarative structure****:* Structure has a subject noun phrase followed by a verb phrase.

  Eg: "*I prefer a morning flight*".

  - ***Subject NP+VP***

- ***Imperative structure***: Structure often begins with a verb phrase, and have no subject. They are called imperative because they are almost always used for commands and suggestions.

  Eg: "*Show the lowest fare*".

  - $S \rightarrow VP$

- ***Yes-no-question structure:*** Structure are often (though not always) used to ask questions (hence the name), and begin with an auxiliary verb, followed by a subject NP, followed by a VP.

  Eg: *"Do any of these flights have stops?".*

  - $S \rightarrow Aux\ NP\ VP$

- ***Wh-question structure:*** These may be broadly grouped into two classes of sentence-level structures.

- The **wh-subject-question** structure is identical to the declarative structure, except that the first noun phrase contains some wh-word.

  Eg: "*Whose flights serve breakfast?*"

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

- In the **wh-non-subject question** structure, the wh-phrase is not the subject of the sentence, and so the sentence includes another subject.
  - In these types of sentences the auxiliary appears before the subject NP, just as in the yes-no-question structures.

  Eg: "What flights do you have from Burbank to Tacoma Washington?"

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

Clauses and Sentences

- ***Clause:*** A clause is the smallest grammatical unit that can express a complete proposition. In traditional grammar, Clauses are often described as forming a complete thought.

- ***Sentences:*** It may form larger structures. **S** may be in the right side of the production

The Noun Phrase

- Noun phrases consisting of a head, the central noun in the noun phrase, along with various modifiers that can occur before or after the head noun. Let's take a close look at the various parts.

- **The Determiners:**
  - Noun phrases can begin with simple lexical determiners

    Examples: a stop, the flights, this flight, those flights, any flights, some flights
  - Determiners are optional in English.
    - For example, determiners may be omitted if the noun they modify is plural:

      Show me *flights* from San Francisco to Denver on weekdays
  - **Mass nouns** also don't require determination
  - substance like ***water*** and ***snow***, don't take the indefinite article "*a*", and don't tend to pluralize.
  - Many abstract nouns are mass nouns (***music, homework***).

Noun phrase

- **The Nominal**
- The nominal construction follows the determiner and contains any pre- and post-head noun modifiers.
- As indicated in grammar L_0, in its simplest form a nominal can consist of a single noun.
  - *Nominal → Noun*

- As we'll see, this rule also provides the basis for the bottom of various recursive rules used to capture more complex nominal constructions.

The Noun Phrase: Before Noun head

- **Predeterminers:**
  - Word classes appearing in the NP before the determiner
    - ***all*** *the flights,* ***all*** *flights*
- **Postdeterminers:**
  - Word classes appearing in the NP between the determiner and the head noun
    - **Cardinal numbers:** *two friends, one stop*
    - **Ordinal numbers:** *the first one, the next day, the second leg, the last flight, the other American flight, and other fares*
    - **Quantifiers:** *many fares*
      - The quantifiers, *much* and *a little* occur only with noncount nouns.

The Noun Phrase: Before Noun head

- Adjectives occur after quantifiers but before nouns.
  - *a **first-class** fare, a **nonstop** flight, the **longest** layover, the **earliest** lunch flight*
- Adjectives can be grouped into a phrase called an **adjective phrase** or **AP.**
  - AP can have an adverb before the adjective
    - *the **least** expensive fare*
- $NP \rightarrow (Det)(Card)(Ord)(Quant)(AP)\ Nominal$

The Noun Phrase: After Noun head

- A head noun can be followed by **postmodifiers**.
- Three kinds of nominal postmodifiers are very common in English:

• Prepositional phrases	all flights ***from Cleveland***
• Non-finite clauses	any flights ***arriving after eleven a.m****.*
• Relative clauses	a flight *that* ***serves breakfast***

The Noun Phrase: After Noun head

- The three most common kinds of **non-finite** postmodifiers are the gerundive (*-ing*), *-ed*, and infinitive form.
  - A gerundive consists of a VP begins with the gerundive (*-ing*)
    - *any of those [leaving on Thursday]*
    - *any flights [arriving after eleven a.m.]*
    - *flights [arriving within thirty minutes of each other]*

$$Nominal \rightarrow Nominal\ GerundVP$$
$$GerundVP \rightarrow GerundV\ NP \mid GerundV\ PP \mid GerundV \mid GerundV\ NP\ PP$$
$$GerundV \rightarrow being \mid preferring \mid ariving \mid leaving \mid \ldots$$

  - Examples of two other common kinds
    - *the last flight* ***to arrive*** *in Boston*
    - *I need to have dinner* ***served***
    - *Which is the aircraft* ***used by this flight?***

The Noun Phrase: After Noun head

- A postnominal relative clause (more correctly a **restrictive relative clause**)
  - is a clause that often begins with a **relative pronoun** (*that* and *who* are the most common).
  - The relative pronoun functions as the subject of the embedded verb,
    - *a flight that serves breakfast*
    - *flights that leave in the morning*
    - *the United flight that arrives in San Jose around ten p.m.*
    - *the one that leaves at ten thirty five*

Nominal $\rightarrow$ *Nominal RelClause*
RelClause $\rightarrow$ *(who | that) VP*

Some grammar rules for English: Coordination

- NPs and other units can be **conjoined** with **coordinations** like *and, or,* and *but.*
  - *Please repeat [$_{NP}$ [$_{NP}$ the flight] and [$_{NP}$ the coast]]*
  - *I need to know [$_{NP}$ [$_{NP}$ the aircraft] and [$_{NP}$ flight number]]*
  - *I would like to fly from Denver stopping in [$_{NP}$ [$_{NP}$ Pittsburgh] and [$_{NP}$ Atlanta]]*
  - $NP \rightarrow NP$ *and* NP
  - $VP \rightarrow VP$ *and* VP
  - $S \rightarrow S$ *and* S

Some grammar rules for English: Verb Phrase and Subcategorization

- The VP consists of the verb and a number of other constituents.

$VP \rightarrow Verb$	disappear
$VP \rightarrow Verb\ NP$	prefer a morning flight
$VP \rightarrow Verb\ NP\ PP$	leave Boston in the morning
$VP \rightarrow Verb\ PP$	leaving on Thursday

- An entire embedded sentence, called **sentential complement**, can follow the verb.

You $[_{VP}$ $[_{V}$ said $[_{S}$ there were two flights that were the cheapest]]]
You $[_{VP}$ $[_{V}$ said $[_{S}$ you had a two hundred sixty six dollar fare]]]
$[_{VP}$ $[_{V}$ Tell] $[_{NP}$ me] $[_{S}$ how to get from the airport in Philadelphia to downtown]]
I $[_{VP}$ $[_{V}$ think $[_{S}$ I would like to take the nine thirty flight]]

$VP \rightarrow Verb\ S$

Some grammar rules for English: Verb Phrase and Subcategorization

- Another potential constituent of the VP is another VP
  - Often the case for verbs like *want, would like, try, intent, need*

I want [$_{VP}$ to fly from Milwaukee to Orlando]
Hi, I want [$_{VP}$ to arrange three flights]
Hello, I'm trying [$_{VP}$ to find a flight that goes from Pittsburgh to Denver after two p.m.]

- Recall that verbs can also be followed by *particles*, word that resemble a preposition but that combine with the verb to form a *phrasal verb*, like *take off*.
  - These particles are generally considered to be an integral part of the verb in a way that other post-verbal elements are not;
  - Phrasal verbs are treated as individual verbs composed of two words.

Some grammar rules for English: Verb Phrase and Subcategorization

- A VP can have many possible kinds of constituents, not every verb is compatible with every VP.
  - *I want a flight ...*
  - *I want to fly to ...*
  - **I found to fly to Dallas.*
- The idea that verbs are compatible with different kinds of complements
  - Traditional grammar **subcategorize** verbs into two categories (transitive and intransitive).
  - Modern grammars distinguish as many as 100 subcategories

Frame	Verb	Example
ϕ	eat, sleep	I want to eat
NP	prefer, **find** leave	Find [$_{NP}$ the flight from Pittsburgh to Boston]
NP NP	show, give, **find**	Show [$_{NP}$ me] [$_{NP}$ airlines with flights from Pittsburgh]
$PP_{from}\, PP_{to}$	fly, travel	I would like to fly [$_{PP}$ from Boston] [$_{PP}$ to Philadelphia]
$NP\ _{PPwith}$	help, load	Can you help [$_{NP}$ me] [$_{PP}$ with a flight]
VPto	prefer, want, need	I would prefer [$_{VPto}$ to go by United airlines]
VPbrst	can, would, might	I can [$_{VPbrst}$ fo from Boston]

Grammar Equivalence and Normal Form

- A formal language is defined as a (possibly infinite) set of strings of words
- Two grammars are equivalent if they generate the same set of strings

- Two kinds of grammar equivalence:
  - **Strong equivalence** - Two grammars are strongly equivalent if they generate the same set of strings and assign the same phrase structure to each sentence (allowing merely renaming of the non-terminal symbols).

  - **Weak equivalence** - Two grammars are weakly equivalent if they generate the same set of strings but do not assign the same phrase structure to each sentence.

Grammar Equivalence and Normal form

- It is sometimes useful to have a **normal form** for grammars, in which each of the productions takes a particular form.

- A context-free grammar is in **Chomsky Normal Form** (CNF) (Chomsky, 1963)
  - if it is ε -free and if in addition each production is either of the form $A \rightarrow B\ C$ or $A \rightarrow a$.

  - That is, the right-hand side of each rule either has **two non-terminal symbols or one terminal symbol.**

Chomsky normal form grammars are **binary branching**, i.e. they have binary trees.

Grammar Equivalence and Normal form

- For example, a rule of the form
  - $A \to B\ C\ D$
- can be converted into the following weakly equivalence CNF rules :
  - $A \to X\ D$
  - $X \to \mathrm{B}\ C$
- Sometimes using binary branching can actually produce smaller grammars.

Parsing with Context Free Grammars

Parsing as Searching

- Unit 1 and unit 2 showed that finding the right path through a finite-state automaton or finding the right transduction for an input, can be viewed as a search problem.

- For finite-state automata, the search is through the space of all possible paths through a machine.

- In syntactic parsing, the parser can be viewed as searching through the space of possible parse trees to find the correct parse tree for a given sentence.

- Just as the search space of possible paths was defined by the structure of an automata, so the search space of possible parse trees is defined by grammar.

- Parse tree are directly useful in applications such as grammar checking in word-processing systems; a sentence which cannot be parsed may have grammatical errors (or at least be hard to read).

- Typically, parse trees serve as an important intermediate stage of representation for semantic analysis, and thus plays an important role in applications like **machine translation**, **question answering** and **information extraction**.

- Parsing algorithms specify how to recognize the strings of a language and assign each string to one (or more) syntactic analyses

What is Parsing?

- The process of taking a string and a grammar and returning all possible parse trees for that string
- That is, find all trees, whose root is the start symbol S, which cover exactly the words in the input

- There are two kinds of constraints that should help for parsing.
  - One set of constraints comes from the data, that is, the input sentence itself. Whatever else is true of the final parse tree, we know that there must be three leaves and they must be the words ***book***, ***that***, and ***flight***.

  - The second kind of constraint comes from the grammar. We know that whatever else is true of the final parse tree, it must have one root, which must be the **start symbol *S***.

Parsing: Example

$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$
$S \rightarrow VP$
$NP \rightarrow Pronoun$
$NP \rightarrow Proper\text{-}Noun$
$NP \rightarrow Det\ Nominal$
$Nominal \rightarrow Noun$
$Nominal \rightarrow Nominal\ Noun$
$Nominal \rightarrow Nominal\ PP$
$VP \rightarrow Verb$
$VP \rightarrow Verb\ NP$
$VP \rightarrow Verb\ NP\ PP$
$VP \rightarrow Verb\ PP$
$VP \rightarrow VP\ PP$
$PP \rightarrow Preposition\ NP$

$Det \rightarrow that \mid this \mid a$
$Noun \rightarrow book \mid flight \mid meal \mid money$
$Verb \rightarrow book \mid include \mid prefer$
$Pronoun \rightarrow I \mid she \mid me$
$Proper\text{-}Noun \rightarrow Houston \mid TWA$
$Aux \rightarrow does$
$Preposition \rightarrow from \mid to \mid on \mid near \mid through$

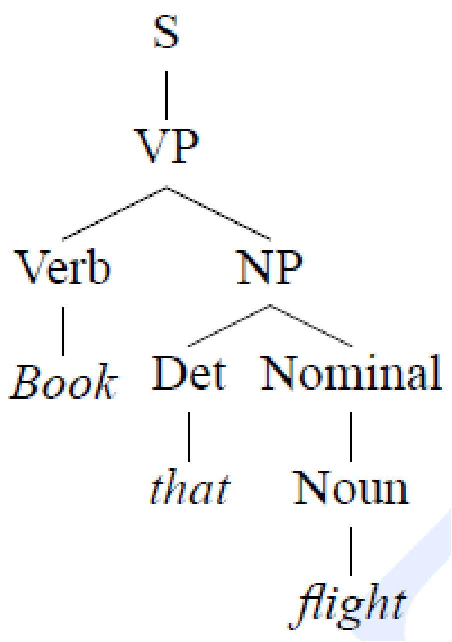


Figure : The $\mathscr{L}_1$ miniature English grammar and lexicon.

Types of Parsing

These two constraints give rise to two search strategies underlying most parsers:

- **Top-down parsing** or **goal-directed search:**
  - Builds from the root S node to the leaves

- **Bottom-up parsing** or **data-directed search**.
  - Parser begins with words of input and builds up trees, applying grammar rules whose RHS matches.

Top-down parsing

- Searches for a parse tree by trying to build from the **root node *S*** down to the **leaves**.

- Let’s consider the search space that a top-down parser explores, assuming for the moment that it builds all possible trees in parallel.

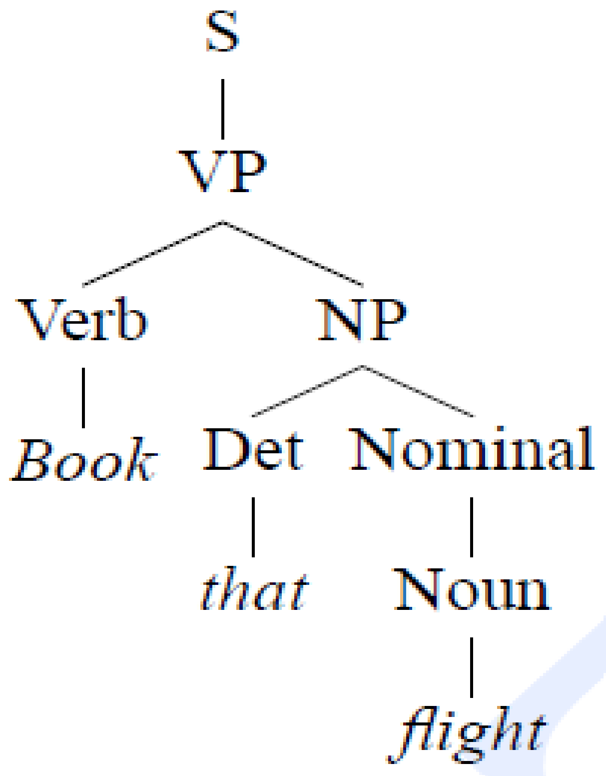
- The algorithm starts by assuming the input can be derived by the designated start symbol *S*.
- The next step is to find the tops of all trees which can start with *S*, by looking for all the grammar rules with *S* on the left-hand side.

- Trees are grown downward until they eventually reach the part-of-speech categories at the bottom of the tree.

- At this point, trees whose leaves fail to match all the words in the input can be rejected

Top-down Parsing Example

$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$
$S \rightarrow VP$
$NP \rightarrow Pronoun$
$NP \rightarrow Proper\text{-}Noun$
$NP \rightarrow Det\ Nominal$
$Nominal \rightarrow Noun$
$Nominal \rightarrow Nominal\ Noun$
$Nominal \rightarrow Nominal\ PP$
$VP \rightarrow Verb$
$VP \rightarrow Verb\ NP$
$VP \rightarrow Verb\ NP\ PP$
$VP \rightarrow Verb\ PP$
$VP \rightarrow VP\ PP$
$PP \rightarrow Preposition\ NP$

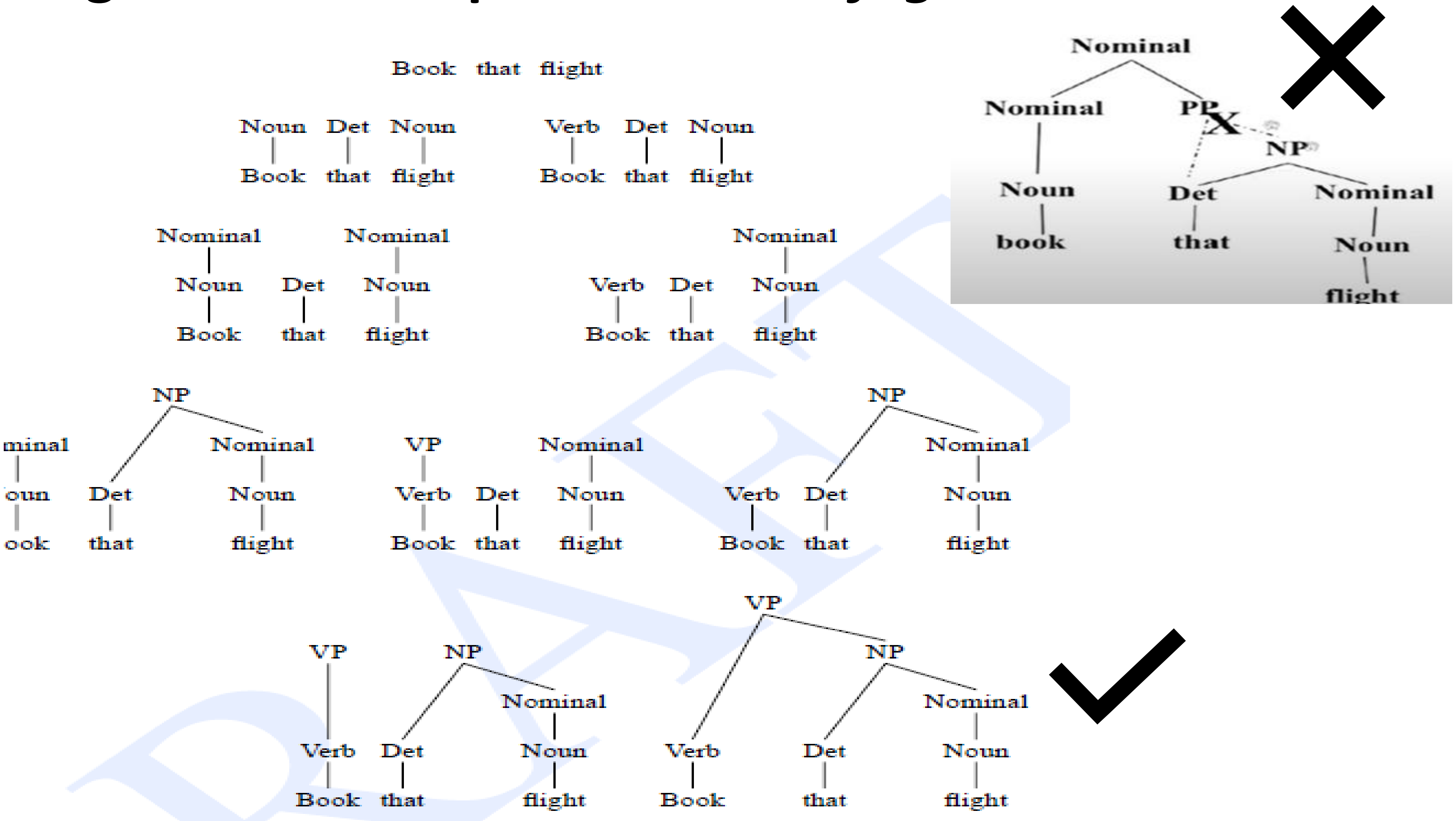


Bottom-up parsing

- In bottom-up parsing the parser starts with the words of the input, and tries to build trees from the words up, again by applying rules from the grammar one at a time.
- The parse is successful if the parser succeeds in building a tree rooted in the start symbol *S* that covers all of the input.

Bottom-up parsing for the example ‘*book that flight*’

$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$
$S \rightarrow VP$
$NP \rightarrow Pronoun$
$NP \rightarrow Proper\text{-}Noun$
$NP \rightarrow Det\ Nominal$
$Nominal \rightarrow Noun$
$Nominal \rightarrow Nominal\ Noun$
$Nominal \rightarrow Nominal\ PP$
$VP \rightarrow Verb$
$VP \rightarrow Verb\ NP$
$VP \rightarrow Verb\ NP\ PP$
$VP \rightarrow Verb\ PP$
$VP \rightarrow VP\ PP$
$PP \rightarrow Preposition\ NP$

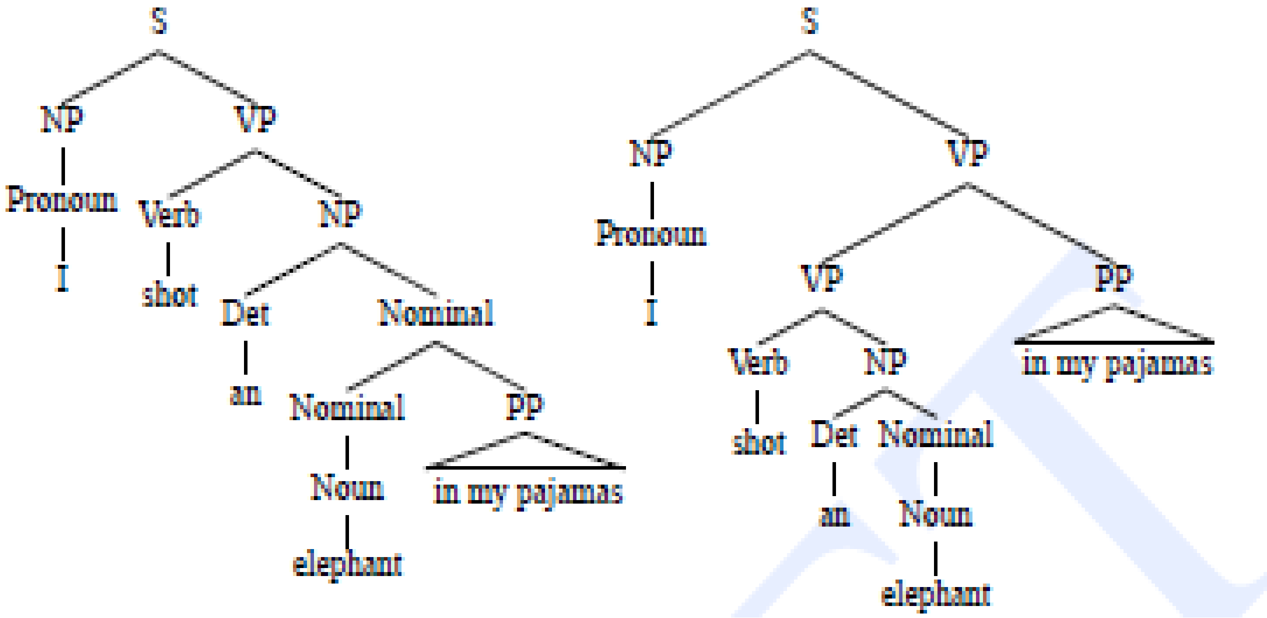


Comparing Top-Down and Bottom-Up Parsing

- The top-down strategy never wastes time exploring trees that cannot result in a full parse tree, since it begins by generating just those trees.
  - But explores many options that never connect to the actual sentence
- In the bottom-up strategy, by contrast, never explores options that do not connect to the actual sentence i.e. no hope of leading to an *S*
  - But can explore options that can never lead to a full parse tree

Ambiguity

- Ambiguity is a serious problem faced by parsers. (Already seen in Part-of-speech ambiguity)
- Ambiguity also arises in the syntactic structures used in parsing and is called as **structural ambiguity**.
  - Structural ambiguity occurs when the grammar assigns more than one possible parse to a sentence.
  - Groucho Marx's well-known line as Captain Spaulding is ambiguous because the phrase *in my pajamas* can be part of the *NP* headed by *elephant* or the verb-phrase headed by *shot*.



- Three common kinds of ambiguity are **attachment ambiguity, coordination ambiguity, and Local ambiguity**.

- A sentence has an **attachment ambiguity** if a particular constituent can be attached to the parse tree at more than one place.
  - The Groucho Marx sentence above is an example of PP-attachment ambiguity.

- In **coordination ambiguity** there are different sets of phrases that can be conjoined by a conjunction like *and*.
  - For example, the phrase *old men and women* can be bracketed as *[old [men and women]]*, referring to *old men* and *old women*, or as *[old men] and [women]*, in which case it is only the men who are old.

- **Local ambiguity** occurs when some part of a sentence is ambiguous, that is, has more than one parse, evenif the whole sentence is not ambiguous.
  - For example the sentence *Book that flight* is unambiguous, but when the parser sees the first word *Book*, it cannot know if it is a verb or a noun until later. Thus it must consider both possible parses

Dynamic Programming Parsing Methods

- **Ambiguity** gives rise to problems and confusions in standard bottom-up or top-down parsers
- Also to avoid extensive repeated work, caching of intermediate results is required
- **Dynamic programming** provides a framework for caching the results and also solving the ambiguity problem, like it helped with the Minimum Edit Distance
  - Recall that dynamic programming approaches systematically fill in tables of solutions to sub-problems.
  - When complete, the tables contain the solution to all the sub-problems needed to solve the problem as a whole

- The efficiency gain arises from the fact that these subtrees are discovered once, stored, and then used in all parses calling for that constituent.
  - This solves the re-parsing problem (subtrees are looked up, not re-parsed)
  - Partially solves the ambiguity problem (the dynamic programming table implicitly stores all possible parses by storing all the constituents with links that enable the parses to be reconstructed)

- Three most widely used methods are
  - The **Cocke-Kasami-Younger (CKY) algorithm**
    - bottom-up parser, requires normalizing the grammar
  - The **Earley algorithm**
    - Top-down parser, doesn't require normalizing the grammar, more complex
  - **Chart Parsing**
    - retain completed phrases in a chart and can combine top-down and bottom-up searches

CKY Parsing (Cocke-Kasami-Younger Parsing)

- Grammars used with it must be in Chomsky Normal Form (CNF)
  - Either, exactly two non-terminals on the RHS
  - Or, 1 terminal symbol on the RHS
- Bottom-up parsing stores phrases formed from all substrings in a triangular table(chart)

Conversion to CNF

- Assuming we're dealing with an ε-free grammar, there are three situations we need to address in any generic grammar:

1. **Rules that mix terminals with non-terminals on the right-hand side**

    - **Solution:** introduce a new dummy non-terminal that covers only the original terminal.
    - **Example**: a rule for an infinitive verb phrase such as INF-VP → to VP would be replaced by the two rules INF-VP → TO VP and TO → to

2. **Rules with a single non-terminal on the right are called unit productions**

    - **Solution:** Unit productions are eliminated by rewriting the right-hand side of the original rules with the right-hand side of all the non-unit production rules that they ultimately lead to.

- **Example:** More formally, if A $\overset{*}{\Rightarrow}$ B by a chain of one or more unit productions, and B→γ is a non-unit production in our grammar, then we add A →γ for each such rule in the grammar, and discard all the intervening unit productions.

3. **Rules with right-hand sides longer than two**

- **Solution:** Introduction of new non-terminals that spread the longer sequences over several new productions.

- **Example:** if we have a rule like A → B Cγ
  - we replace the leftmost pair of non-terminals with a new non-terminal and introduce a new production result in the following new rules.
    - X1 → B C
    - A → X1 γ

Conversion to CNF : Example

Original Grammar

S → NP VP
S → Aux NP VP
S → VP
NP → Pronoun
NP → Proper-Noun
NP → Det Nominal
Nominal → Noun
Nominal → Nominal Noun
Nominal → Nominal PP
VP → Verb
VP → Verb NP
VP → VP PP
PP → Prep NP
Pronoun → I | he | she | me
Noun → book | flight | meal | money
Verb → book | include | prefer
Proper-Noun → Houston | NWA

Chomsky Normal Form

S → NP VP
S → X1 VP
X1 → Aux NP
S → book | include | prefer
S → Verb NP
S → VP PP
NP → I | he | she | me
NP → Houston | NWA
NP → Det Nominal
Nominal → book | flight | meal | money
Nominal → Nominal Noun
Nominal → Nominal PP
VP → book | include | prefer
VP → Verb NP
VP → VP PP
PP → Prep NP
Pronoun → I | he | she | me
Noun → book | flight | meal | money
Verb → book | include | prefer
Proper-Noun → Houston | NWA

The entire conversion process can be summarized as follows:

1. **Copy all conforming rules to the new grammar unchanged**
2. **Convert terminals within rules to dummy non-terminals**
3. **Convert unit-productions**
4. **Binarize all rules and add to new grammar**

CKY Parsing

- Involves parsing substrings of length 1, then length 2, and so on until the entire string has been parsed.
- This is useful because the shorter substrings from previous iterations can be used when applying grammar rules to parse the longer substrings.

- Let n be the number of words in the input. Think about $n + 1$ lines separating them, numbered 0 to n.
- x_{ij} will denote the words between line i and j
- We build a table so that x_{ij} contains all the possible non-terminal spanning for words between line i and j.
- We build the Table bottom-up.

- For a sentence of length ***n***, we will work with the upper-triangular portion of an **(*n*+1)×(*n*+1)** matrix.
  - Each **cell [*i, j*]** in this matrix contains a set of non-terminals that represent all the constituents that span positions ***i*** through ***j*** of the input

CKY parsing for CFG

a 1	pilot 2	likes 3	flying 4	planes 5

$S \longrightarrow NP\ VP$
$VP \longrightarrow VBG\ NNS$
$VP \longrightarrow VBZ\ VP$
$VP \longrightarrow VBZ\ NP$
$NP \longrightarrow DT\ NN$
$NP \longrightarrow JJ\ NNS$
$DT \longrightarrow a$
$NN \longrightarrow pilot$
$VBZ \longrightarrow likes$
$VBG \longrightarrow flying$
$JJ \longrightarrow flying$
$NNS \longrightarrow planes$

a 1	pilot 2	likes 3	flying 4	planes 5
DT				

$S \rightarrow NP\ VP$
$VP \rightarrow VBG\ NNS$
$VP \rightarrow VBZ\ VP$
$VP \rightarrow VBZ\ NP$
$NP \rightarrow DT\ NN$
$NP \rightarrow JJ\ NNS$
$DT \rightarrow a$
$NN \rightarrow pilot$
$VBZ \rightarrow likes$
$VBG \rightarrow flying$
$JJ \rightarrow flying$
$NNS \rightarrow planes$

Step 1

a 1	pilot 2	likes 3	flying 4	planes 5
DT				
	NN			

$S \rightarrow NP\ VP$
$VP \rightarrow VBG\ NNS$
$VP \rightarrow VBZ\ VP$
$VP \rightarrow VBZ\ NP$
$NP \rightarrow DT\ NN$
$NP \rightarrow JJ\ NNS$
$DT \rightarrow a$
$NN \rightarrow pilot$
$VBZ \rightarrow likes$
$VBG \rightarrow flying$
$JJ \rightarrow flying$
$NNS \rightarrow planes$

Step 2

a 1	pilot 2	likes 3	flying 4	planes 5
DT	NP			
	NN			

$S \rightarrow NP\ VP$
$VP \rightarrow VBG\ NNS$
$VP \rightarrow VBZ\ VP$
$VP \rightarrow VBZ\ NP$
$NP \rightarrow DT\ NN$
$NP \rightarrow JJ\ NNS$
$DT \rightarrow a$
$NN \rightarrow pilot$
$VBZ \rightarrow likes$
$VBG \rightarrow flying$
$JJ \rightarrow flying$
$NNS \rightarrow planes$

Step 3

a 1	pilot 2	likes 3	flying 4	planes 5
DT	NP			
	NN			
		VBZ		

$S \rightarrow NP\ VP$
$VP \rightarrow VBG\ NNS$
$VP \rightarrow VBZ\ VP$
$VP \rightarrow VBZ\ NP$
$NP \rightarrow DT\ NN$
$NP \rightarrow JJ\ NNS$
$DT \rightarrow a$
$NN \rightarrow pilot$
$VBZ \rightarrow likes$
$VBG \rightarrow flying$
$JJ \rightarrow flying$
$NNS \rightarrow planes$

Step 4

a 1	pilot 2	likes 3	flying 4	planes 5
DT	NP			
	NN	-		
		VBZ		

S → *NP VP*
VP → *VBG NNS*
VP → *VBZ VP*
VP → *VBZ NP*
NP → *DT NN*
NP → *JJ NNS*
DT → *a*
NN → *pilot*
VBZ → *likes*
VBG → *flying*
JJ → *flying*
NNS → *planes*

Step 5

a 1	pilot 2	likes 3	flying 4	planes 5
DT	NP	-		
	NN	-		
		VBZ		

S → *NP VP*
VP → *VBG NNS*
VP → *VBZ VP*
VP → *VBZ NP*
NP → *DT NN*
NP → *JJ NNS*
DT → *a*
NN → *pilot*
VBZ → *likes*
VBG → *flying*
JJ → *flying*
NNS → *planes*

Step 6

a 1	pilot 2	likes 3	flying 4	planes 5
DT	NP	-		
	NN	-		
		VBZ		
			JJ	

#

S → *NP VP*
VP → *VBG NNS*
VP → *VBZ VP*
VP → *VBZ NP*
NP → *DT NN*
NP → *JJ NNS*
DT → *a*
NN → *pilot*
VBZ → *likes*
VBG → *flying*
JJ → *flying*
NNS → *planes*

Step 7

a 1	pilot 2	likes 3	flying 4	planes 5
DT	NP	-		
	NN	-		
		VBZ		
			JJ VBG	

#

S → *NP VP*
VP → *VBG NNS*
VP → *VBZ VP*
VP → *VBZ NP*
NP → *DT NN*
NP → *JJ NNS*
DT → *a*
NN → *pilot*
VBZ → *likes*
VBG → *flying*
JJ → *flying*
NNS → *planes*

Step 8

a 1	pilot 2	likes 3	flying 4	planes 5
DT	NP	-	-	
	NN	-	-	
		VBZ	-	
			JJ VBG	

S → *NP VP*
VP → *VBG NNS*
VP → *VBZ VP*
VP → *VBZ NP*
NP → *DT NN*
NP → *JJ NNS*
DT → *a*
NN → *pilot*
VBZ → *likes*
VBG → *flying*
JJ → *flying*
NNS → *planes*

Step 9

a 1	pilot 2	likes 3	flying 4	planes 5
DT	NP	-	-	
	NN	-	-	
		VBZ	-	
			JJ VBG	
				NNS

S → *NP VP*
VP → *VBG NNS*
VP → *VBZ VP*
VP → *VBZ NP*
NP → *DT NN*
NP → *JJ NNS*
DT → *a*
NN → *pilot*
VBZ → *likes*
VBG → *flying*
JJ → *flying*
NNS → *planes*

Step 10

a 1	pilot 2	likes 3	flying 4	planes 5
DT	NP	-	-	
	NN	-	-	
		VBZ	-	
			JJ VBG	NP
				NNS

S → *NP VP*
VP → *VBG NNS*
VP → *VBZ VP*
VP → *VBZ NP*
NP → *DT NN*
NP → *JJ NNS*
DT → *a*
NN → *pilot*
VBZ → *likes*
VBG → *flying*
JJ → *flying*
NNS → *planes*

Step 11

a 1	pilot 2	likes 3	flying 4	planes 5
DT	NP	-	-	
	NN	-	-	
		VBZ	-	
			JJ VBG	NP VP
				NNS

S → *NP VP*
VP → *VBG NNS*
VP → *VBZ VP*
VP → *VBZ NP*
NP → *DT NN*
NP → *JJ NNS*
DT → *a*
NN → *pilot*
VBZ → *likes*
VBG → *flying*
JJ → *flying*
NNS → *planes*

Step 12

a 1	pilot 2	likes 3	flying 4	planes 5
DT	NP	-	-	S
	NN	-	-	-
		VBZ	-	VP VP
			JJ VBG	NP VP
				NNS

#

$S \rightarrow NP\ VP$
$VP \rightarrow VBG\ NNS$
$VP \rightarrow VBZ\ VP$
$VP \rightarrow VBZ\ NP$
$NP \rightarrow DT\ NN$
$NP \rightarrow JJ\ NNS$
$DT \rightarrow a$
$NN \rightarrow pilot$
$VBZ \rightarrow likes$
$VBG \rightarrow flying$
$JJ \rightarrow flying$
$NNS \rightarrow planes$

Step13

a 1	pilot 2	likes 3	flying 4	planes 5
DT	NP	-	-	S S
	NN	-	-	-
		VBZ	-	VP VP
			JJ VBG	NP VP
				NNS

$S \rightarrow NP\ VP$
$VP \rightarrow VBG\ NNS$
$VP \rightarrow VBZ\ VP$
$VP \rightarrow VBZ\ NP$
$NP \rightarrow DT\ NN$
$NP \rightarrow JJ\ NNS$
$DT \rightarrow a$
$NN \rightarrow pilot$
$VBZ \rightarrow likes$
$VBG \rightarrow flying$
$JJ \rightarrow flying$
$NNS \rightarrow planes$

Step 14

CKY Algorithm

function CKY-PARSE(*words, grammar*) **returns** *table*

for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**
    $table[j-1, j] \leftarrow \{A \mid A \rightarrow words[j] \in grammar\}$
    for $i \leftarrow$ **from** $j-2$ **downto** 0 **do**
        for $k \leftarrow i+1$ **to** $j-1$ **do**
            $table[i,j] \leftarrow table[i,j] \cup$
                $\{A \mid A \rightarrow BC \in grammar,$
                    $B \in table[i,k],$
                    $C \in table[k,j]\}$

STATISTICAL PARSING

- The CKY algorithm could represent some ambiguities but were not equipped to resolve them.

- A probabilistic parser offers a solution to the problem: compute the probability of each interpretation, and choose the most-probable interpretations.

- The most commonly used probabilistic grammar is the **probabilistic context-free grammar** (PCFG), a probabilistic augmentation of context-free grammars in which each rule is associated with a probability

PROBABILISTIC CONTEXT-FREE GRAMMARS

- PCFG differs from CFG by augmenting each rule in *R* with a conditional probability:

  $$A \rightarrow b \; [p]$$

- Here *p* expresses the probability that the given non-terminal *A* will be expanded to the sequence b. That is, *p* is the conditional probability of a given expansion b given the left-hand-side (LHS) non-terminal *A*.

- We can represent this probability as ***P*(*A*→b)** or as ***P*(*A*→b|*A*)** or as **P(RHS|LHS)**

- Thus if we consider all the possible expansions of a non-terminal, the sum of their probabilities must be 1

$$\sum_{\beta} P(A \rightarrow \beta) = 1$$

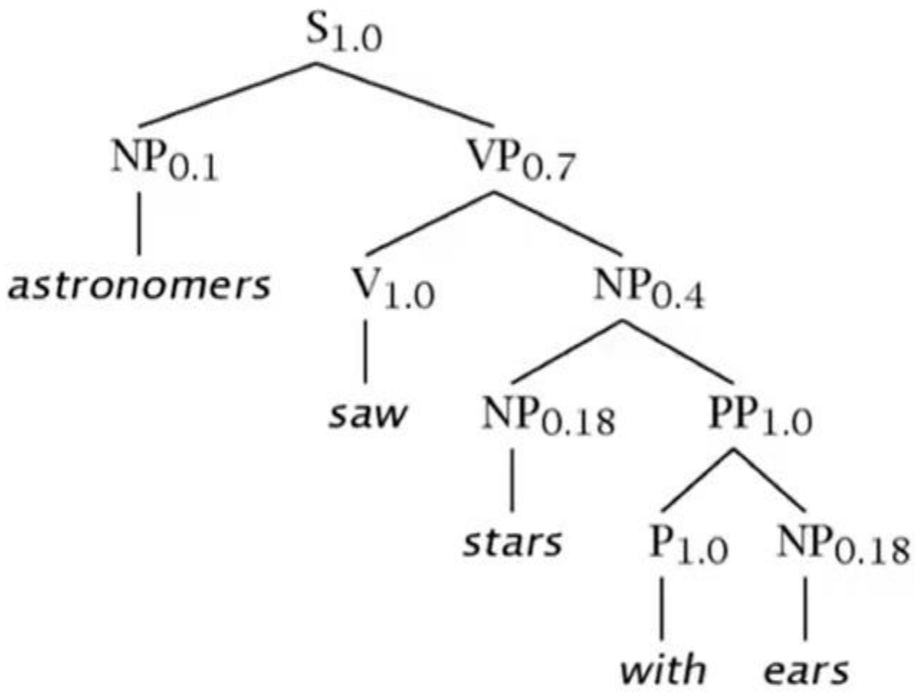
A Simple PFGS (in CNF)

S	→	NP VP	1.0
VP	→	V NP	0.7
VP	→	VP PP	0.3
PP	→	P NP	1.0
P	→	*with*	1.0
V	→	*saw*	1.0

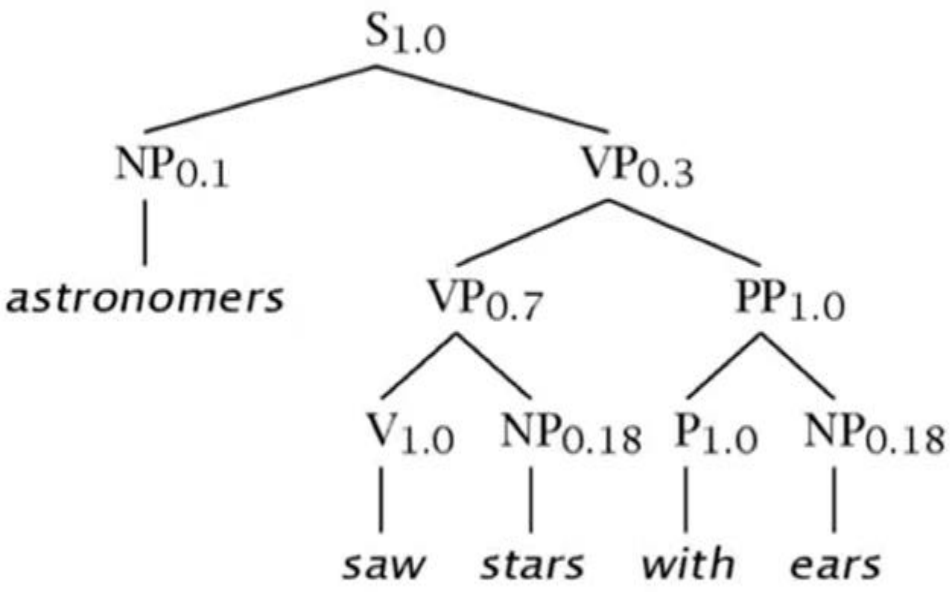
NP	→	NP PP	0.4
NP	→	*astronomers*	0.1
NP	→	*ears*	0.18
NP	→	*saw*	0.04
NP	→	*stars*	0.18
NP	→	*telescope*	0.1

Example trees

t_1:



t_2:



Probabilities of Trees and Strings

- $P(t)$: The probability of tree is the product of the probabilities of the rules used to generate it
- $P(w_{1n})$: The probability of the string is the sum of the probabilities of the trees which have that string as their yield

Probabilities of Trees and Strings

w_{15} = *astronomers saw stars with ears*

$P(t_1) = 1.0 * 0.1 * 0.7 * 1.0 * 0.4 * 0.18 * 1.0 * 1.0 * 0.18 = 0.0009072$

$P(t_2) = 1.0 * 0.1 * 0.3 * 0.7 * 1.0 * 0.18 * 1.0 * 1.0 * 0.18 = 0.0006804$

$P(w_{15}) = P(t_1) + P(t_2) = 0.0009072 + 0.0006804 = 0.0015876$

Probabilities

- Parse tree 1: $.05 \times .20 \times .30 \times .20 \times .60 \times .20 \times .75 \times .10 \times .30 = 1.62 \times 10^{-6}$
- Parse tree 2: $.05 \times .05 \times .30 \times .20 \times .60 \times .75 \times .10 \times .15 \times .75 \times .30 = 2.28 \times 10^{-7}$

Features of PCFG

- As the number of possible trees for a given input grows, a PCFG gives some idea of the plausibility of a particular parse
- *But* the probability estimates are based purely on structural factors, and do not factor in lexical co-occurrence. Thus, PCFG does not give a very good idea of the plausibility of the sentence.
- Real text tends to have grammatical mistakes. PCFG avoids this problem by ruling out nothing, but by giving implausible sentences a low probability
- In practice, a PCFG is a worse language model for English than an n-gram model
- All else being equal, the probability of a smaller tree is greater than a larger tree

CKY for PCFG

a 1	pilot 2	likes 3	flying 4	planes 5

$S \longrightarrow NP\ VP$	[1.0]
$VP \longrightarrow VBG\ NNS$	[0.1]
$VP \longrightarrow VBZ\ VP$	[0.1]
$VP \longrightarrow VBZ\ NP$	[0.3]
$NP \longrightarrow DT\ NN$	[0.3]
$NP \longrightarrow JJ\ NNS$	[0.4]
$DT \longrightarrow a$	[0.3]
$NN \longrightarrow pilot$	[0.1]
$VBZ \longrightarrow likes$	[0.4]
$VBG \longrightarrow flying$	[0.5]
$JJ \longrightarrow flying$	[0.1]
$NNS \longrightarrow planes$	[.34]

CKY for PCFG

0.3 * 0.1 * 0.3

a 1	pilot 2	likes 3	flying 4	planes 5
DT [0.3]	NP [.009]	-	-	S [$1.4688 x 10^{-5}$] S [$6.12 x 10^{-6}$]
	NN [0.1]	-	-	-
		VBZ [0.4]	-	VP [.001632] VP [.00068]
			JJ [0.1] VBG [0.5]	NP [.0136] VP [.017]
				NNS [.34]

0.4 * 0.0136* 0.3

0.4 * 0.017 * 0.1

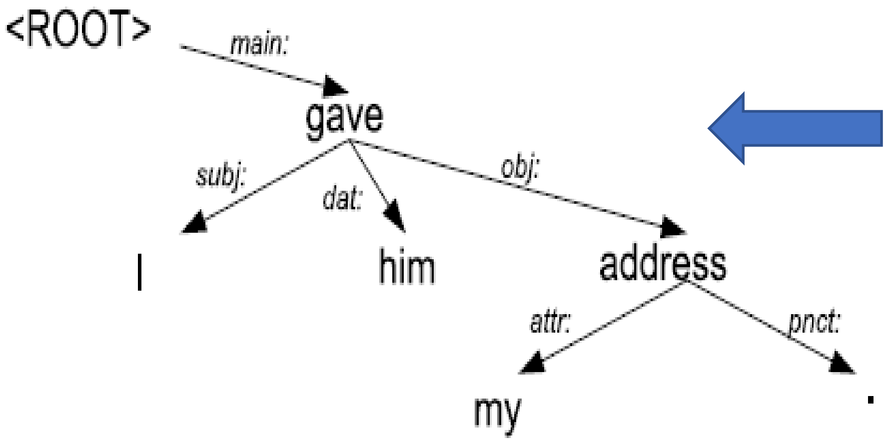
Dependency Grammars

- **Dependency grammars** that are becoming quite important in speech and language processing, where constituents and phrase-structure rules do not play any fundamental role.

- Instead of that, the syntactic structure of a sentence is described purely in terms of words and binary semantic or syntactic relations between these words.

- It follows the notion of traditional grammar "**parsing a sentence into subject and predicate**" that is based on lexical relations rather than constituent relations.

Dependency Grammars



Shows an example parse of the sentence I gave him my address, using the dependency grammar formalism.

Note that there are no non-terminal or phrasal nodes; each link in the parse tree holds between two lexical nodes (augmented with the special <ROOT> node).

The links are drawn from a fixed inventory of around 35 relations, most of which roughly represent grammatical functions or very general semantic relations.

Dependency	Description
subj	syntactic subject
obj	direct object (incl. sentential complements)
dat	indirect object
pcomp	complement of a preposition
comp	predicate nominals (complements of copulas)
tmp	temporal adverbials
loc	location adverbials
attr	premodifying (attributive) nominals (genitives, etc.)
mod	nominal postmodifiers (prepositional phrases, etc.)

Dependency Grammars

- Advantage of dependency formalisms is the strong predictive parsing power that words have for their dependents.

- Knowing the identity of the verb is often a very useful cue for deciding which noun is likely to be the subject or the object.

- Another advantage of pure dependency grammars is their ability to handle languages with relatively **free word order**.
  - For example an *object* might occur before or after a *location adverbial* or a **comp**.

- A phrase-structure grammar would need a separate rule for each possible place in the parse tree such that an adverbial phrase could occur.
- A dependency grammar would just have one link-type representing this particular adverbial relation. Thus a dependency grammar abstracts away from word-order variation, representing only the information that is necessary for the parse.

- Convert the following CFG to CNF
    - S->bA|aB
    - A->bAA|aS|a
    - B->aBB|bS|b

- Step1: List the productions which are already in CNF

  A->a
  B->b

- Step2:Replace the terminals on the right by the new non terminals
  - S->bA
    - ✓ S->CA
    - ✓ C->b
  - S->aB
    - ✓ S->DB
    - ✓ D->a
  - ▪ A->aS
    - ✓ A->DS
    - ✓ D->a
  - B->bS
    - B->CS
    - C->b

- A->bAA
  - ✓ A->CAA
  - ✓ C->b
- B->aBB
  - ✓ B->DBB
  - ✓ D->a

- Step3: According to CNF RHS must contain only 2 non terminals. So, convert such productions.
  - A->CAA
    - ✓ A->EA
    - ✓ E->CA
  - B->DBB
    - ✓ B->FB
    - ✓ F->DB

- Step4: Check for productions with only one non terminal in RHS

  No such productions in the given problem.

The final CFG in CNF is:

- ✓ S->CA
- ✓ C->b
- ✓ S->DB
- ✓ D->a
- ✓ A->DS
- ✓ D->a
- ✓ B->CS
- ✓ A->EA
- ✓ E->CA
- ✓ B->FB
- ✓ F->DB

S->CA|DB

C->b

D->a

A->DS|EA|a

B->CS|FB|b

E->CA

F->DB

Exercise

1. Show the CYK Algorithm with the following example:
   - CNF grammar G
     - S→ AB | BC
     - A→ BA | a
     - B→ CC | b
     - C→ AB | a
   - w is **ababa**
   - Whether **ababa** is in **L(G)**?

2. Use the CYK algorithm to find the parse tree for “ Book the flight through Houston” using the CNF form shown below.

$S \rightarrow NP\ VP$	$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$	$S \rightarrow X1\ VP$
	$X1 \rightarrow Aux\ NP$
$S \rightarrow VP$	$S \rightarrow book \mid include \mid prefer$
	$S \rightarrow Verb\ NP$
	$S \rightarrow X2\ PP$
	$S \rightarrow Verb\ PP$
	$S \rightarrow VP\ PP$
$NP \rightarrow Pronoun$	$NP \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$NP \rightarrow TWA \mid Houston$
$NP \rightarrow Det\ Nominal$	$NP \rightarrow Det\ Nominal$
$Nominal \rightarrow Noun$	$Nominal \rightarrow book \mid flight \mid meal \mid money$
$Nominal \rightarrow Nominal\ Noun$	$Nominal \rightarrow Nominal\ Noun$
$Nominal \rightarrow Nominal\ PP$	$Nominal \rightarrow Nominal\ PP$
$VP \rightarrow Verb$	$VP \rightarrow book \mid include \mid prefer$
$VP \rightarrow Verb\ NP$	$VP \rightarrow Verb\ NP$
$VP \rightarrow Verb\ NP\ PP$	$VP \rightarrow X2\ PP$
	$X2 \rightarrow Verb\ NP$
$VP \rightarrow Verb\ PP$	$VP \rightarrow Verb\ PP$
$VP \rightarrow VP\ PP$	$VP \rightarrow VP\ PP$
$PP \rightarrow Preposition\ NP$	$PP \rightarrow Preposition\ NP$

3. Consider grammar L

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

Let w = baaba. Is the word is in grammar L?

END