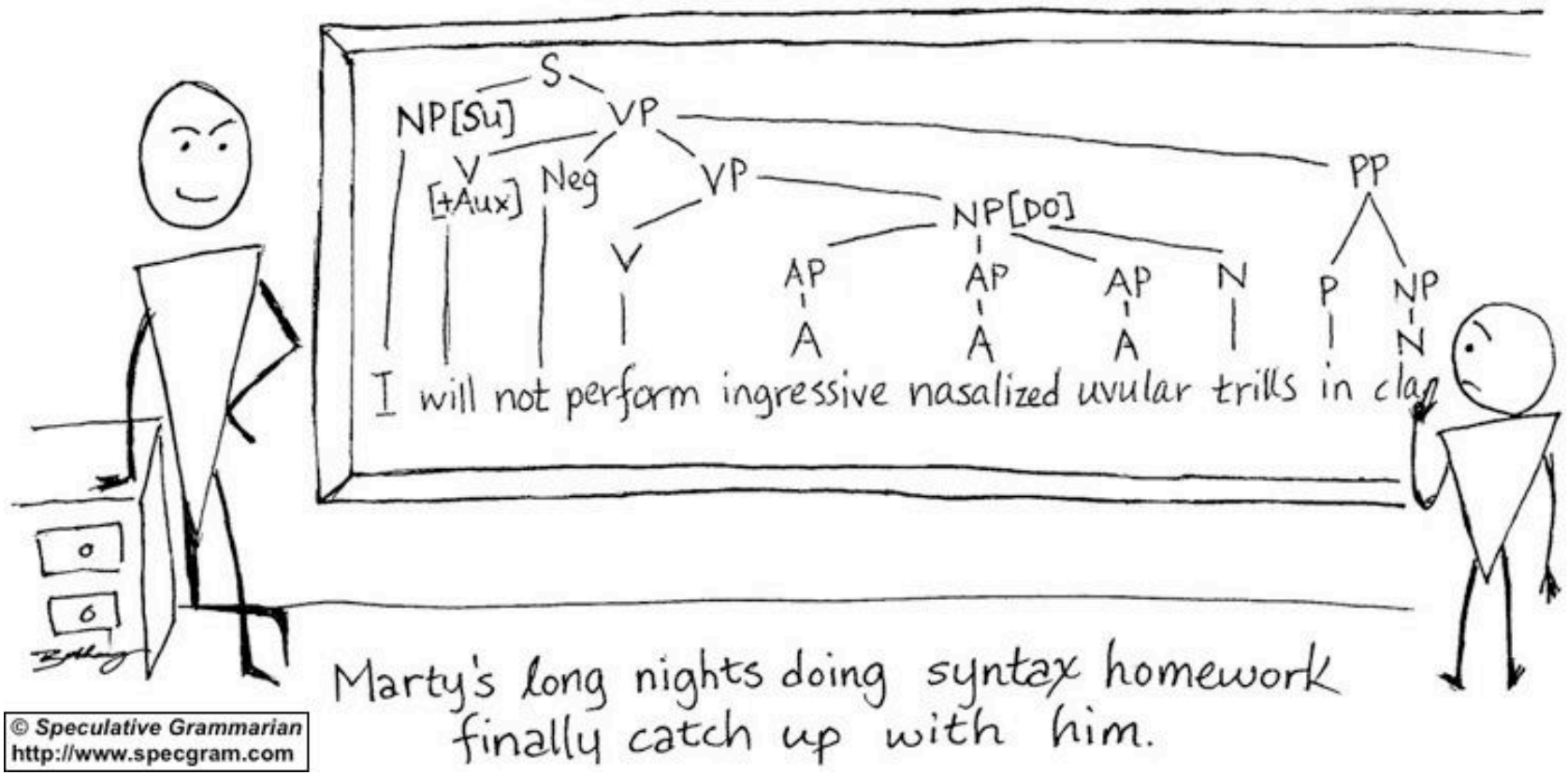
BBM 495

Context Free Grammars (CFG) PARSING

LECTURER: BURCU CAN

2019-2020 SPRING



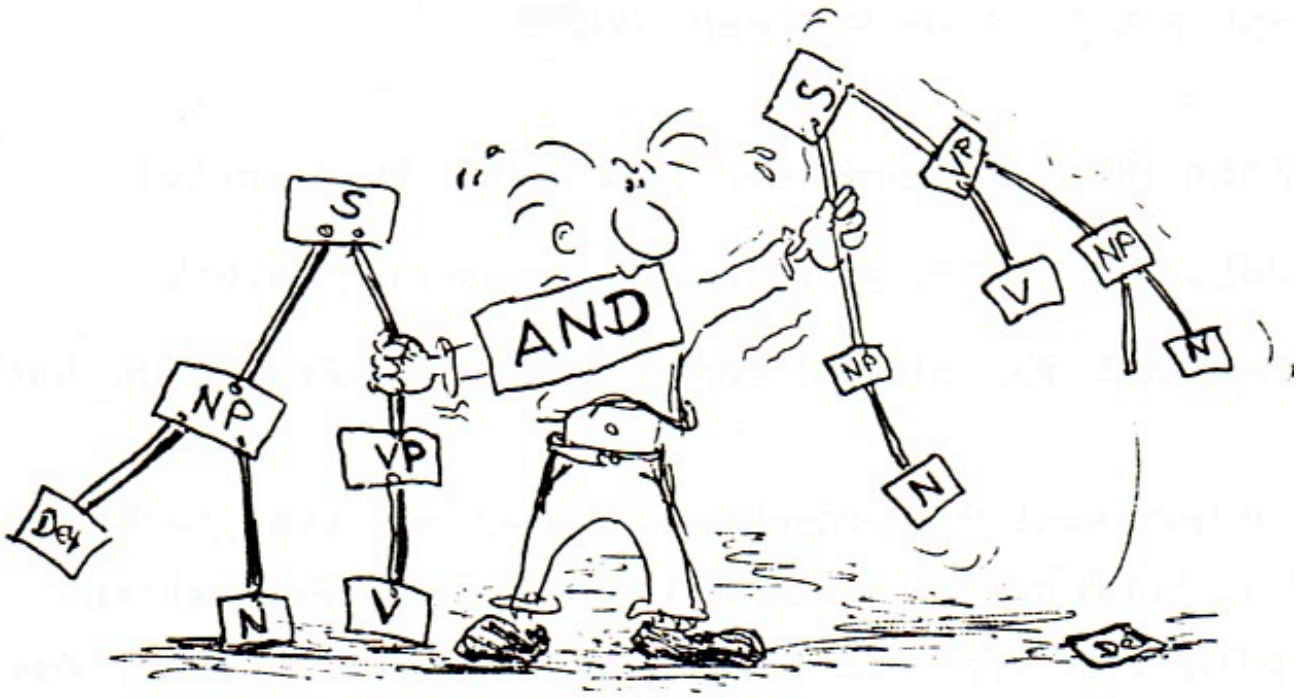
Marty's long nights doing syntax homework finally catch up with him.

- Up to now, we dealt with words:
  - POS tagging, morphological analysis, spell checking

- From now on, we will be dealing with sentences:
  - Syntactic parsing and semantic analysis
  - But this week, only syntactic parsing…

Syntax

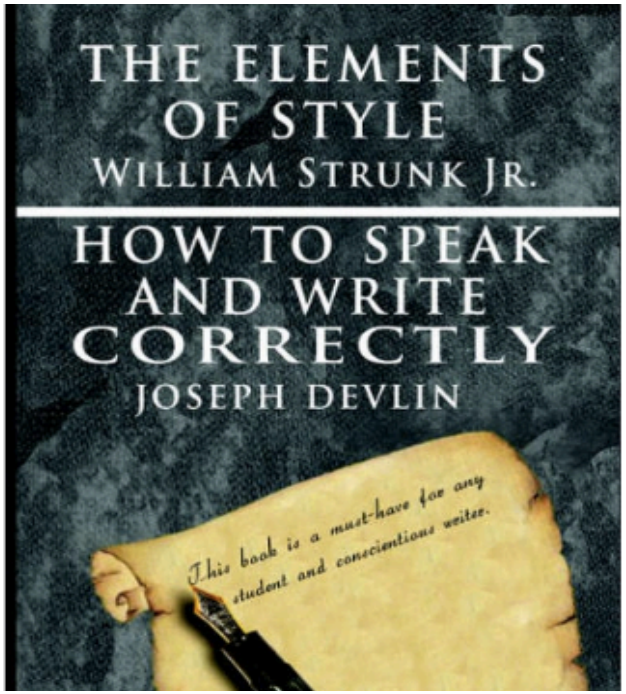
- Syntax (of natural languages) describe how words are strung together to form components of sentences, and how those components are strung together to form sentences.

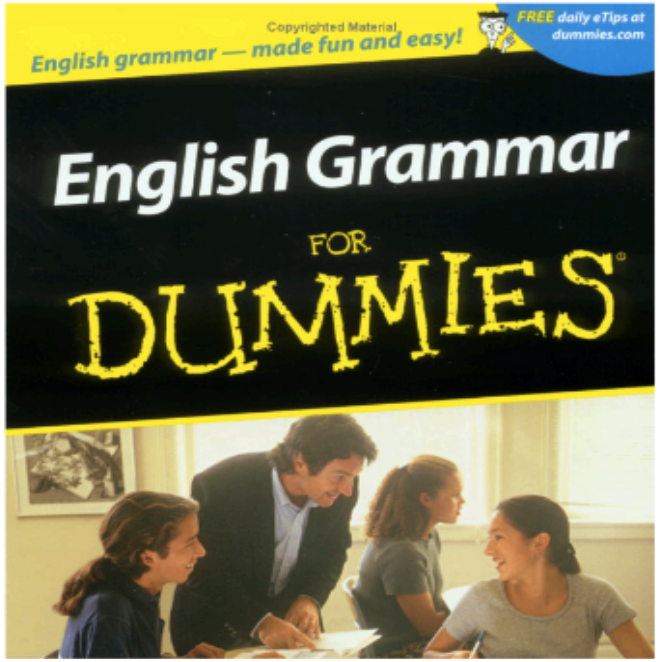


Some Applications of Syntax

- We may need knowledge of syntax in the following applications
  - Grammar checkers
  - Question answering
  - Information extraction
  - Generation
  - Translation
  - Understanding

What is Grammar?

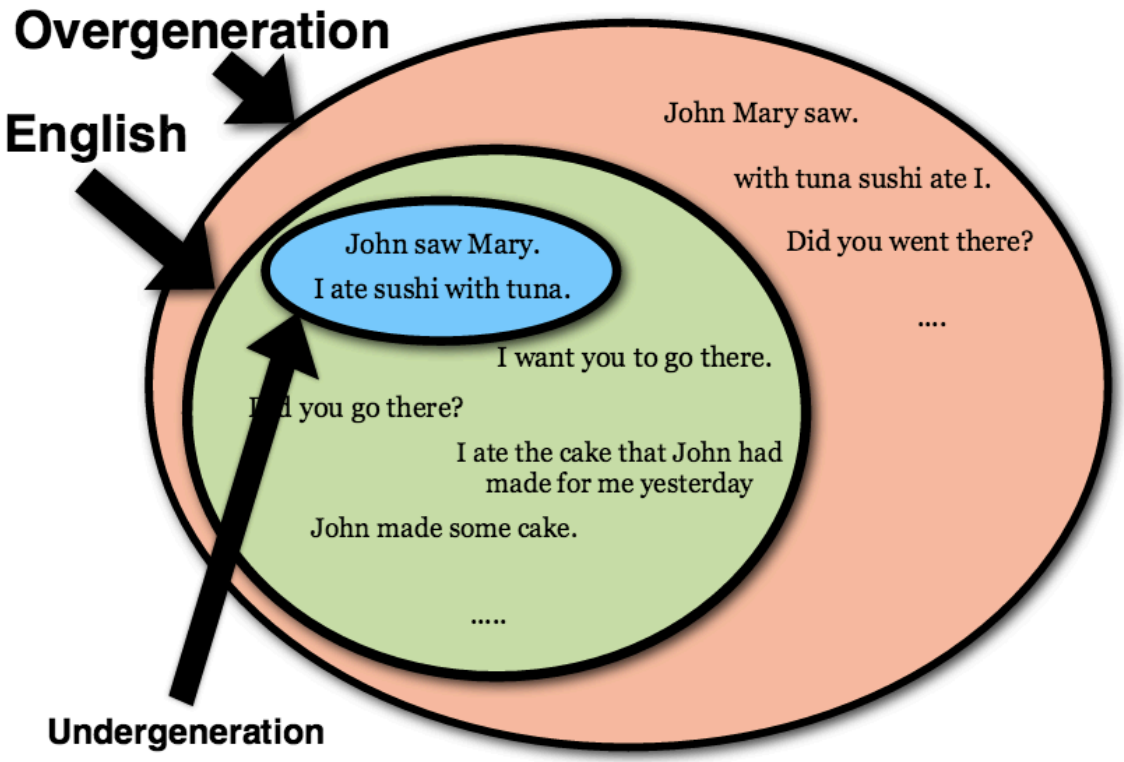




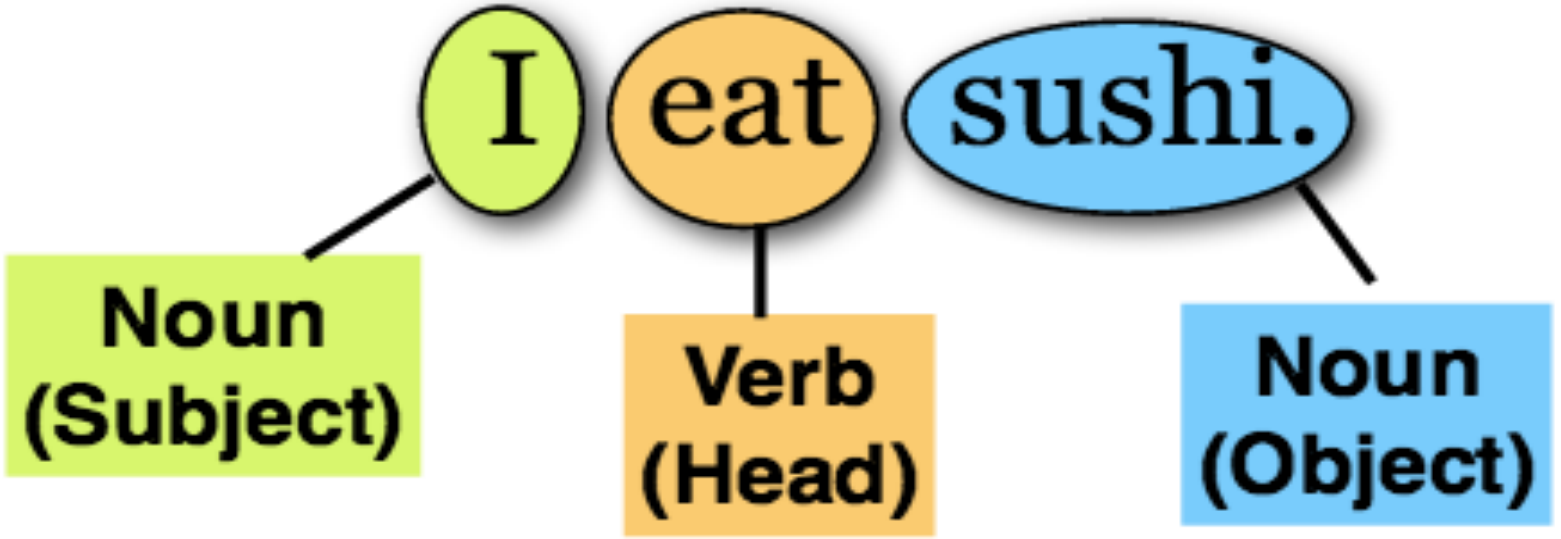
Why not Regular Grammars?

- Regular grammars are too weak to capture the syntactic structures in the natural language.
  - Regular grammars do not handle dependencies

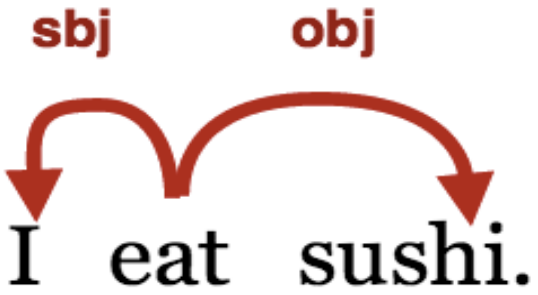
- Can we define a program that generates all English sentences?
  - The number of sentences is infinite.
  - But we need our program to be finite.

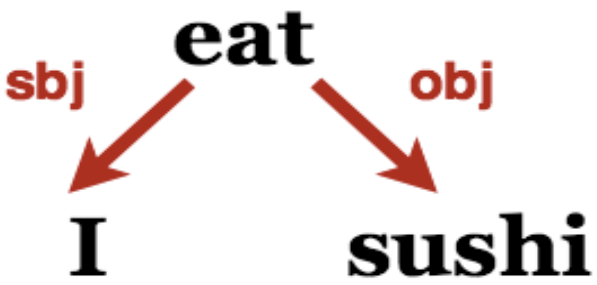


Basic sentence structure



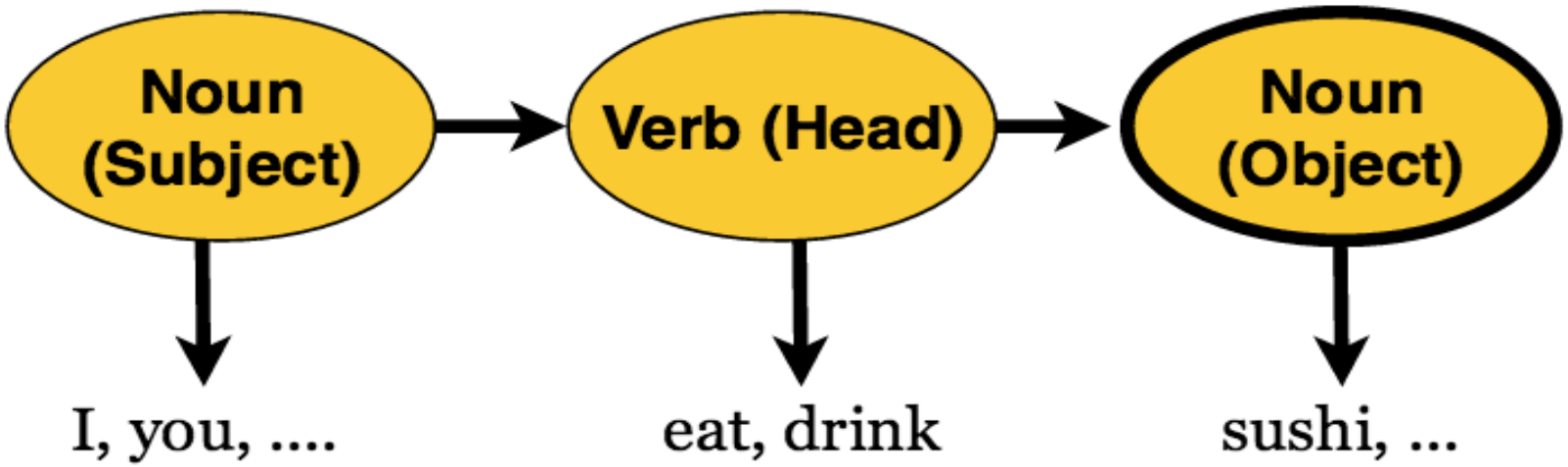
Dependency Graph





A Finite State Automaton (FSA)

A Hidden Markov Model



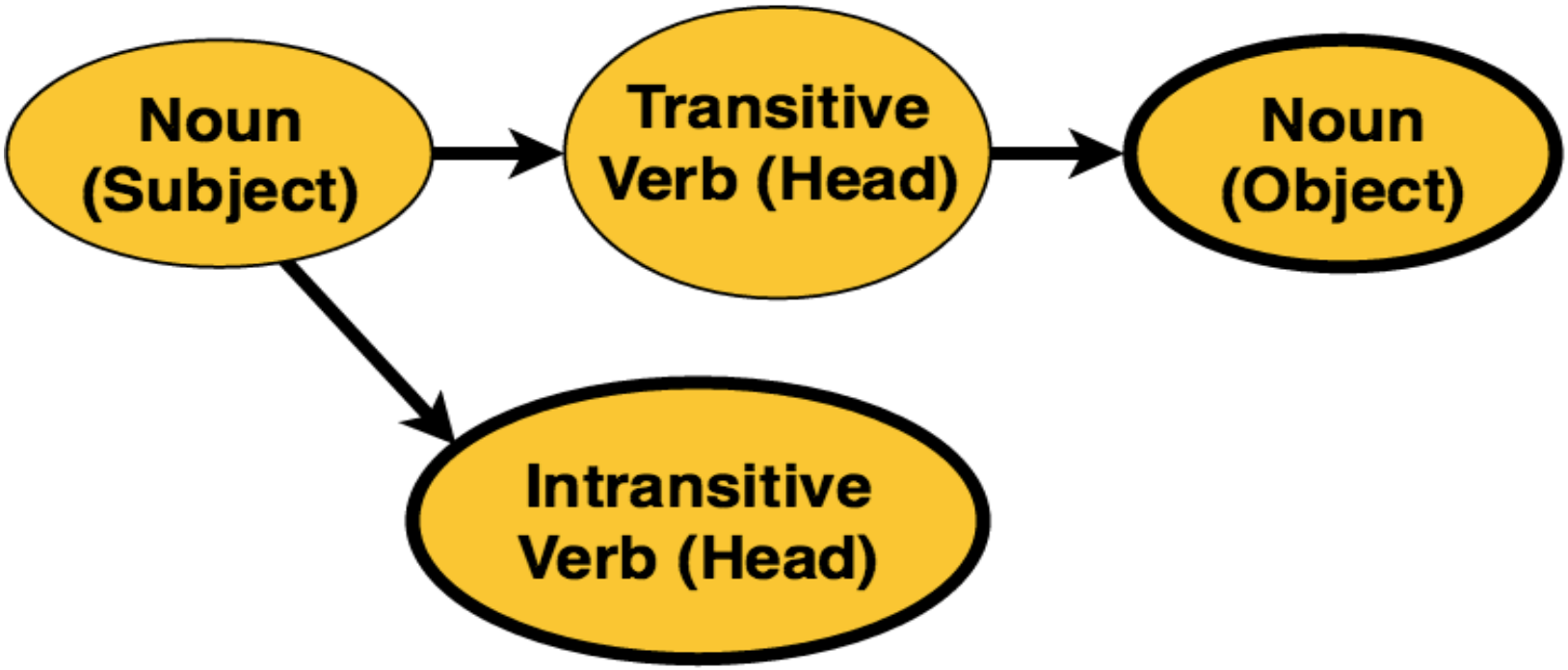
Words take arguments

- I eat sushi. (Correct)
- I eat sushi you. (???)
- I sleep sushi. (???)
- I give sushi. (???)
- I drink sushi. (?)

  - **Intransitive verbs** (sleep) take only one subject.
  - **Transitive verbs** (eat) take also one (direct) object.
  - **Ditransitive verbs** (give) take also one (indirect) object.

  Selectional preferences: The object of *eat* should be edible.

A Better FSA

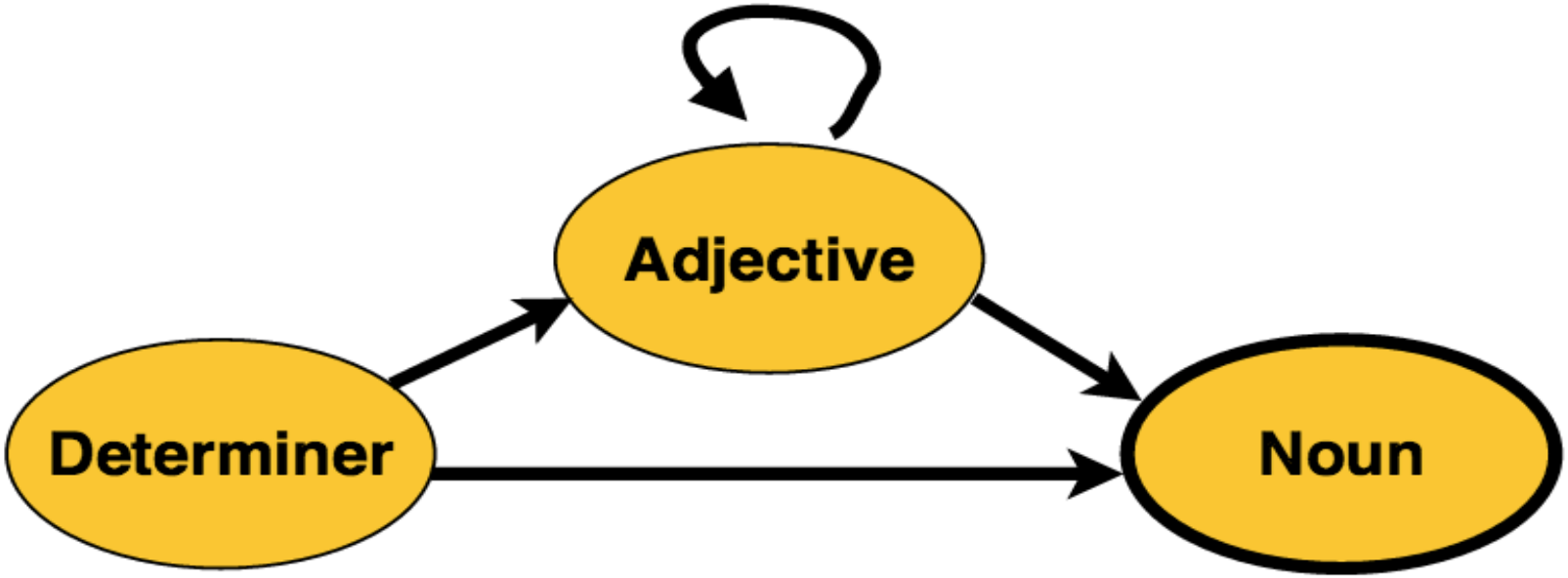

Noun
(Subject)
Transitive
Verb (Head)
Noun
(Object)
Intransitive
Verb (Head)

Language is recursive

- the ball
- the big ball
- the big, red ball
- the big, red, heavy ball
- …

- Adjectives can modify nouns.
- The number of modifiers a word can have is unlimited (in theory).

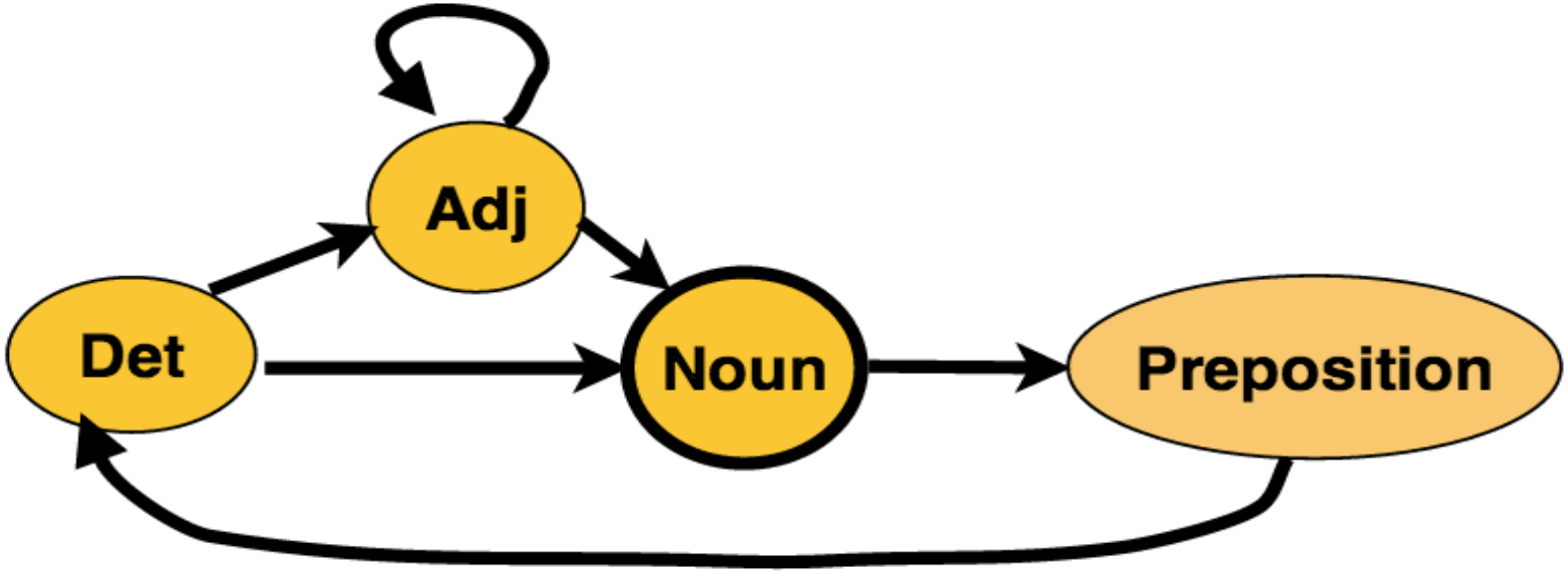
Another FSA



Recursion can be more complex

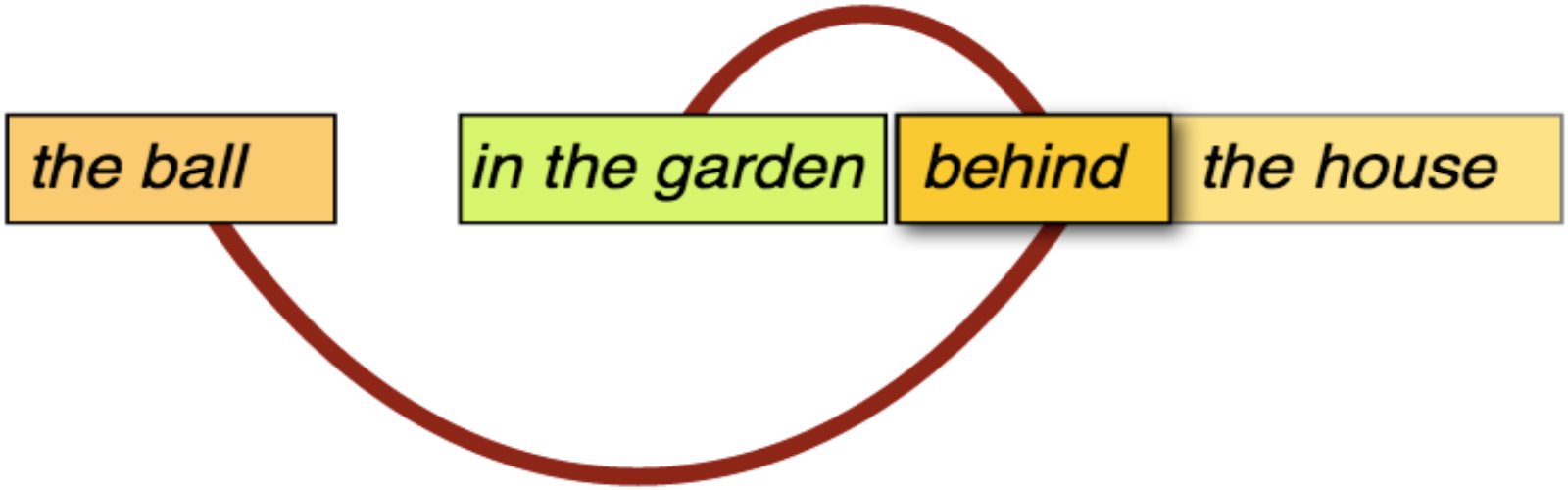
- the ball
- the ball in the garden
- the ball in the garden behind the house
- the ball in the garden behind the house next to school
- …

Another FSA

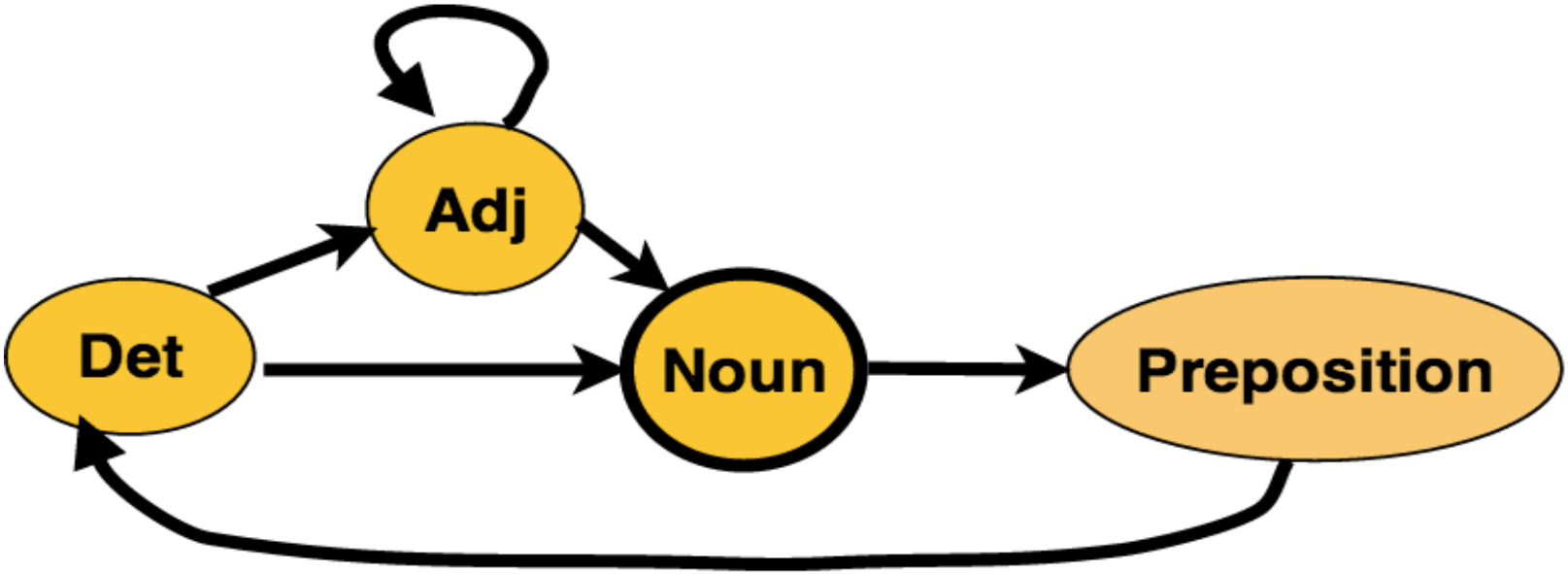


So, why do we need anything beyond *regular grammars*?

What does this mean?



FSA does not generate hierarchical structure



What is the structure of a sentence?

- Sentence structure is hierarchical:
  - A sentence consists of words (I, eat, sushi, with, tuna)
  - …which form phrases or constituents (sushi with tuna)

Sentence structure defines dependencies between words or phrases:

Context Free Grammars

Capture recursion

- Language has complex constituents
  - (“the garden behind the house”)

- CFGs define nonterminal categories to capture equivalent constituents.

Context-free grammars

- A CFG is a 4-tuple $< N, \sum, R, S >$ consisting of:

- A set of nonterminals N
  - (e.g. N={S, NP, VP, PP, Noun, Verb, …})
- A set of terminals $\sum$
  - (e.g. $\sum$ ={I, you, he, eat, drink, sushi, ball, …})
- A set of rules R
  - $R \subseteq \{A \rightarrow \beta$ with left-hand-side $A \in N$
  - and right-hand-side $\beta \in (N \cup \sum)^*\}$

- A start symbol $S \in N$

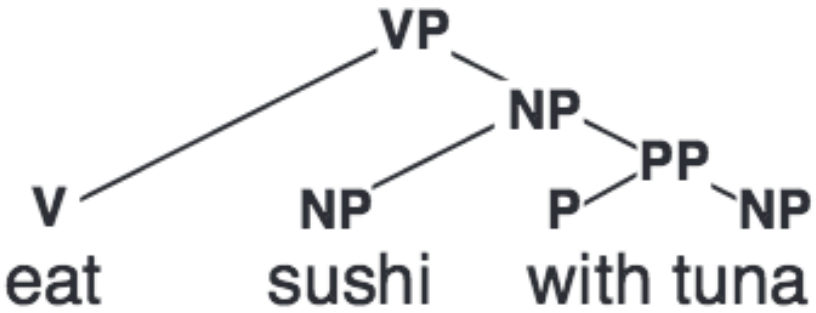
An example

- DT → {the, a}
- N → {ball, garden, house, sushi}
- P → {in, behind, with}
- NP → DT N
- NP → NP PP
- PP → P NP

- N: noun
- P: preposition
- NP: noun phrase
- PP: prepositional phrase

CFGs define parse trees

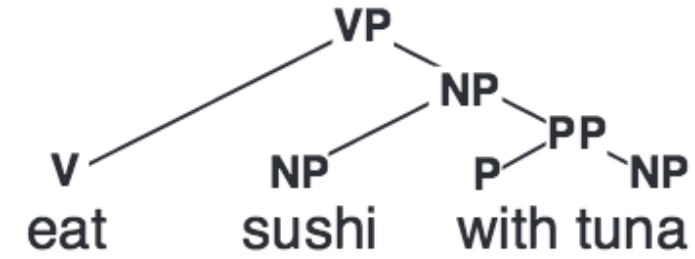
- N→ {sushi, tuna}
- P → {with}
- V → {eat}
- NP → N
- NP → NP PP
- PP → P NP
- VP → V NP

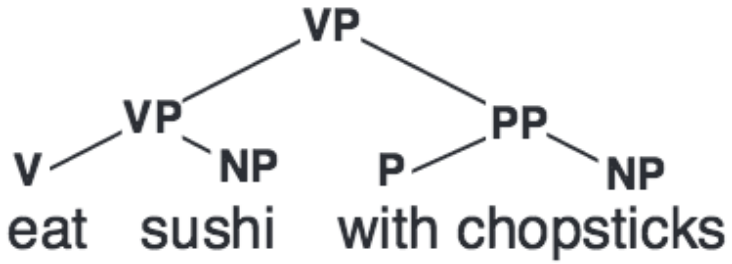


Defining grammars for natural language

- There are two ways to represent the structure:

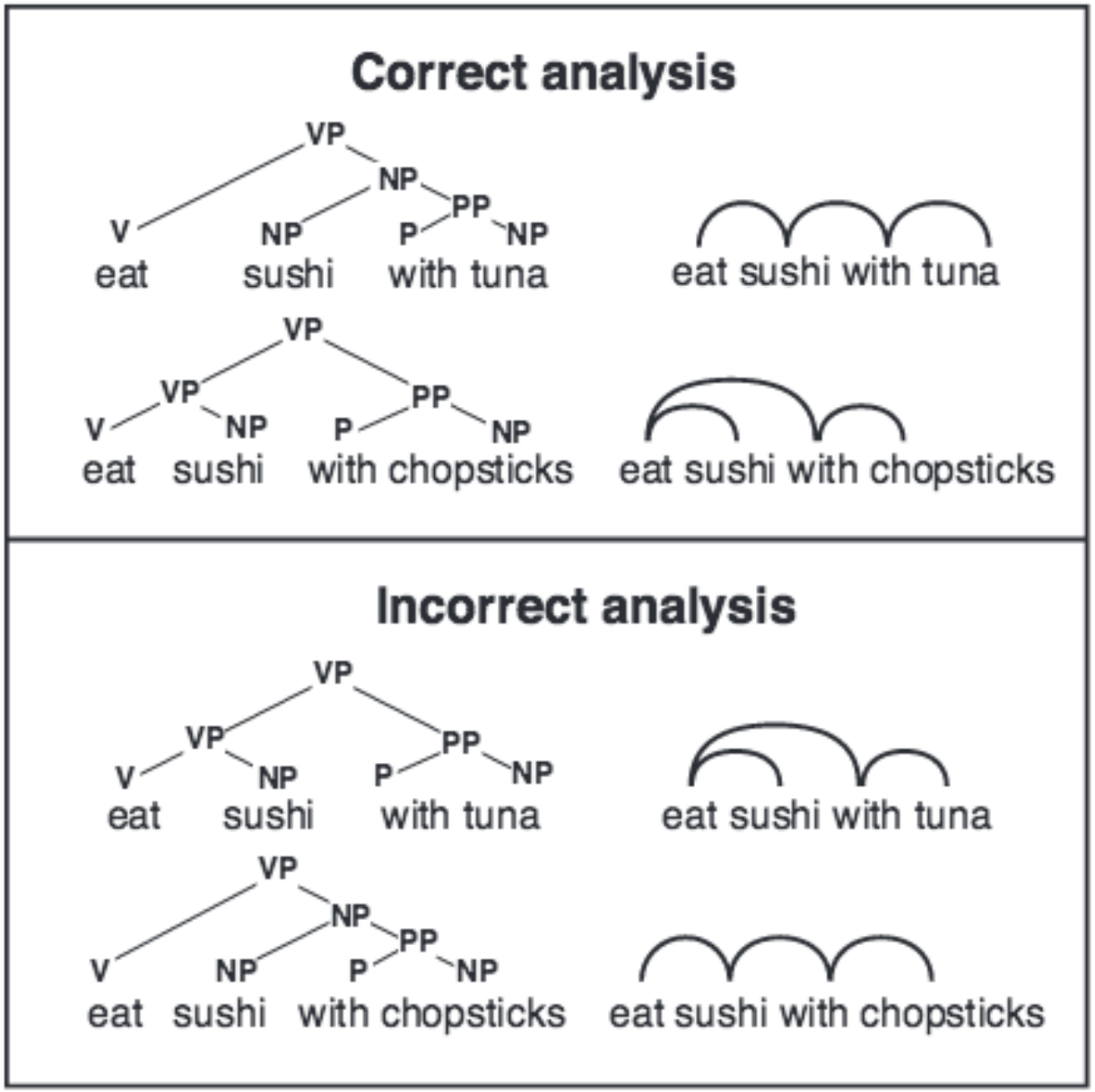
Phrase structure trees





Dependency trees

Structure (syntax) corresponds to meaning (semantics)



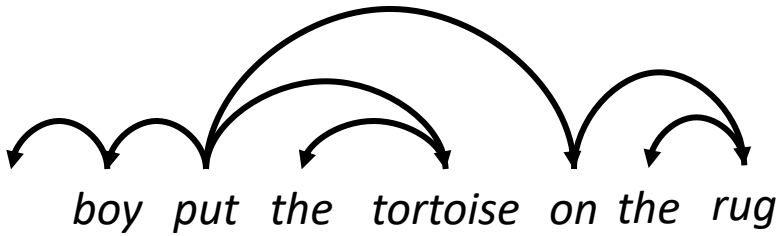
Dependency Grammar

- Dependency Grammars (DGs) describe the structure of sentences as graph.
  - The **nodes** of the graph are the **words**.
  - The **edges** of the graph are the **dependencies**.

- Note: the relationship between DG and CFGs:
  - If a CFG phrase structure tree is translated into DG, the resulting dependency graph has no crossing edges.

Dependency structure

- Dependency structure shows which words depend on (modify or are arguments of) which other words.



Developing grammars

- We must do a lot to develop grammars for natural languages.
- Here we look at some constituents (syntactic substructures) in natural languages.
- The key constituents are:
  - Sentences
  - Noun Phrases
  - Verb Phrases
  - Prepositional Phrases

Sentence Types

- Declarative Sentences
  - S → NP VP — He left
- Imperative Sentences
  - S → VP — Get out!
- Yes-No Questions
  - S → Aux NP VP — Did you decide?
- WH-Questions
  - S → WH-Word Aux NP VP — What did you decide?

Noun phrases (NP)

- Simple NPs:
- [He] sleeps. (pronoun)
- [John] sleeps. (proper name)
- [A student] sleeps. (determiner+noun)

- Complex NPs:
- [A tall student] sleeps. (det+adj+noun)
- [The student in the back] sleeps. (NP+PP)
- [The student who likes MTV] sleeps. (NP+relative clause)

The NP fragment

- NP→ Pronoun
- NP→ ProperName
- NP→ Det Noun
- NP→ NP RelClause

- Det→{a,the,every}
- Pronoun→{he,she,…}
- ProperName→{John,Mary,…}
- Noun→{book, ball, ….}

Adjective phrases (AdjP)
Prepositional phrases (PP)

- AdjP → Adj
- AdjP → Adv AdjP
- Adj → {big, small, red, …}
- Adv → {very, really, …}

- PP →P NP
- P→{with, in, above, …}

The Verb Phrase (VP)

- He [eats].
- He [eats sushi].
- He [gives John sushi].
- He [eats sushi with chopsticks].

- VP→ V
- VP → V NP
- VP → V NP PP
- VP → VP PP

- V → {eats, sleeps, gives, …}

Sentences

- [He eats sushi].
- [Sometimes, he eats sushi].
- [In Japan, he eats sushi].

- S→ NP VP
- S → AdvP S
- S→ PP S

- He says [he eats sushi].
- VP → V_{comp} S
- V_{comp}→ {says, think, believes}

Sentences redefined

- [He eats sushi]. (correct)
- [I eats sushi]. ???
- [They eats sushi]. ???

- S → $NP_{3sg}VP_{3sg}$
- S → $NP_{1sg}VP_{1sg}$
- S → $NP_{3pl}VP_{3pl}$

- We need features to capture agreement (number, case, person, etc).

PARSING

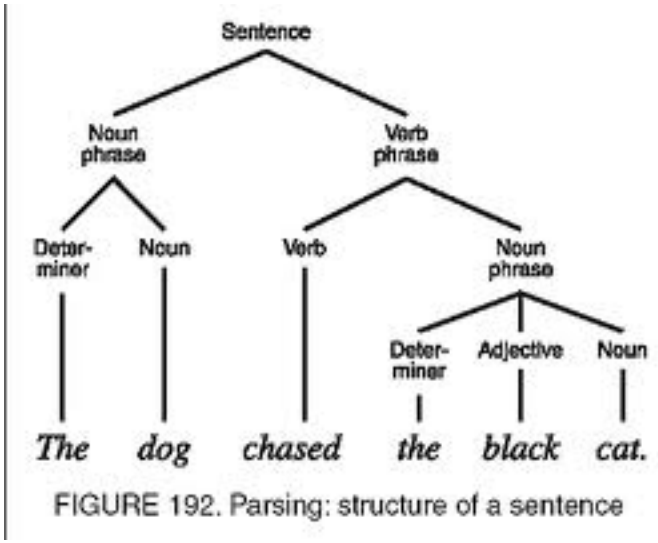


FIGURE 192. Parsing: structure of a sentence

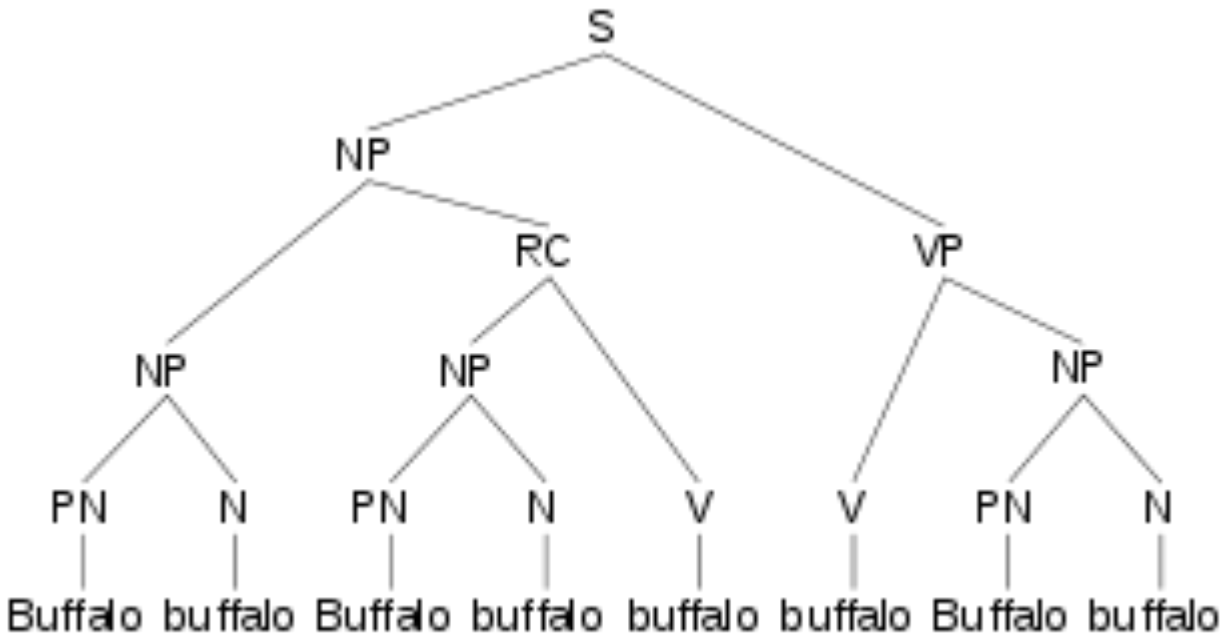
Why do we need parsing?

- Parse trees are directly useful in applications such as grammar checking in word processing systems.

- Also has an important role in
  - Semantic analysis
  - Machine translation
  - Question answering
  - Information extraction, etc.

Syntactic parsing

- Örnek: Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo



- Diğer buffalo bizonlarının korkuttuğu bufalo bizonları, yine aynı buffalo bizonlarını korkutmaktadır

Example

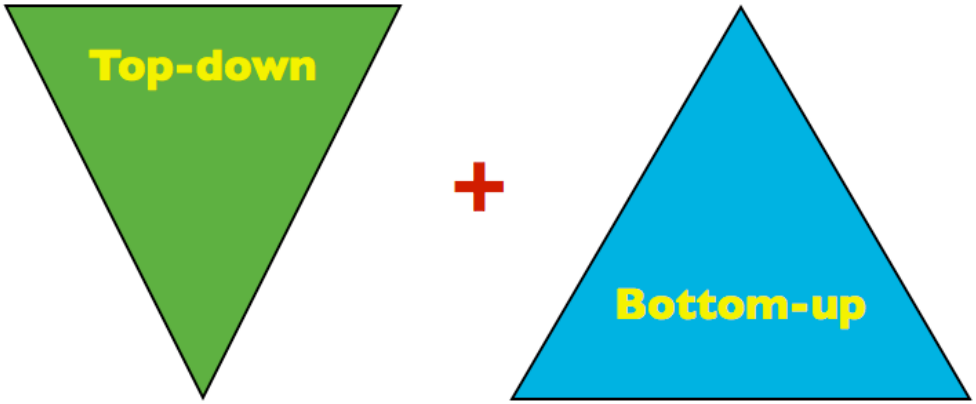
- In question answering:
- What books were written by British women authors before 1800?
- The subject of the sentence is ‘what books’

Parsing

- Parsing with a CFG is the task of assigning a correct parse tree (or derivation) to a string.
- The correct means that it is consistent with the input and grammar. It doesn't mean that it's the "right" tree in global sense of correctness.
- The leaves of the parse tree cover all and only the input, and that parse tree corresponds to a valid derivation according to the grammar.
- The parsing can be viewed as a search. The search space corresponds to the space of parse trees generated by the grammar. The search is guided by the structure of space and by the input.

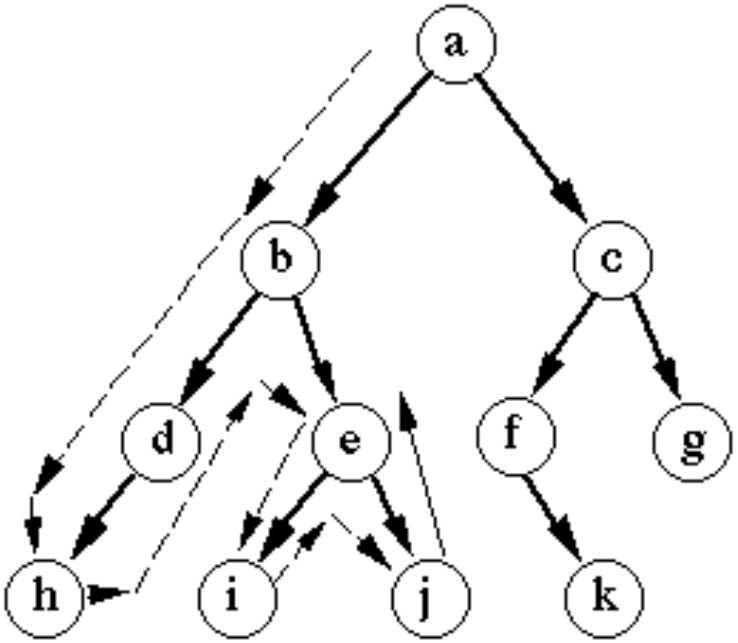
Parsing

- Parsing can be done in two ways:
- Top-down parsing: from the start symbol down.
- Bottom-up parsing: from the symbols up.

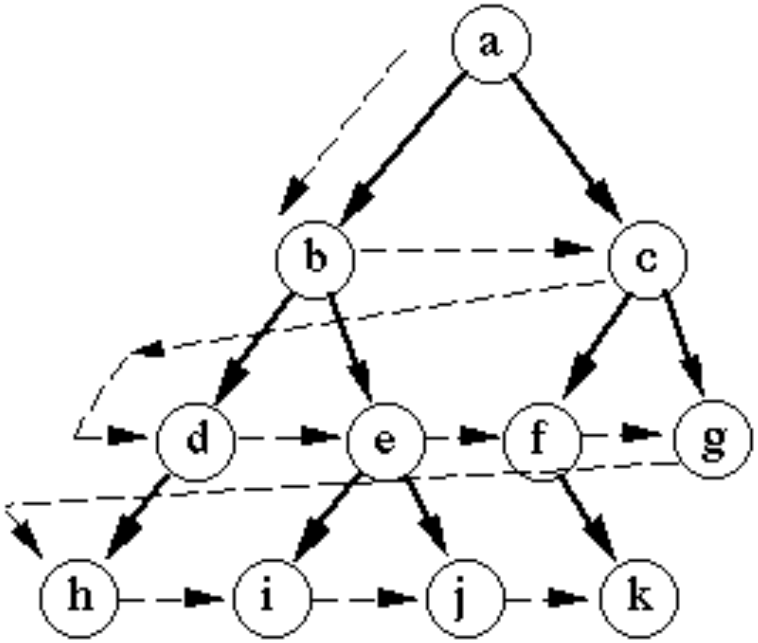


Basic Top-Down Parsing

- A top-down parser searches a parse tree by trying to build from the root node S (start symbol) down to leaves.
- First, we create the root node, then we create its children. We chose one of its children and then we create its children.
- We can search the search space of the parse trees:
  - breadth first search -- level by level search
  - depth first search -- first we search one of the children



Depth-first search



Breadth-first search

A Simple English Grammar

S → NP VP
S → Aux NP VP
S → VP

NP → Det NOM
NP → ProperNoun

NOM → Noun
NOM → Noun NOM
NOM → NOM PP

VP → Verb
VP → Verb NP

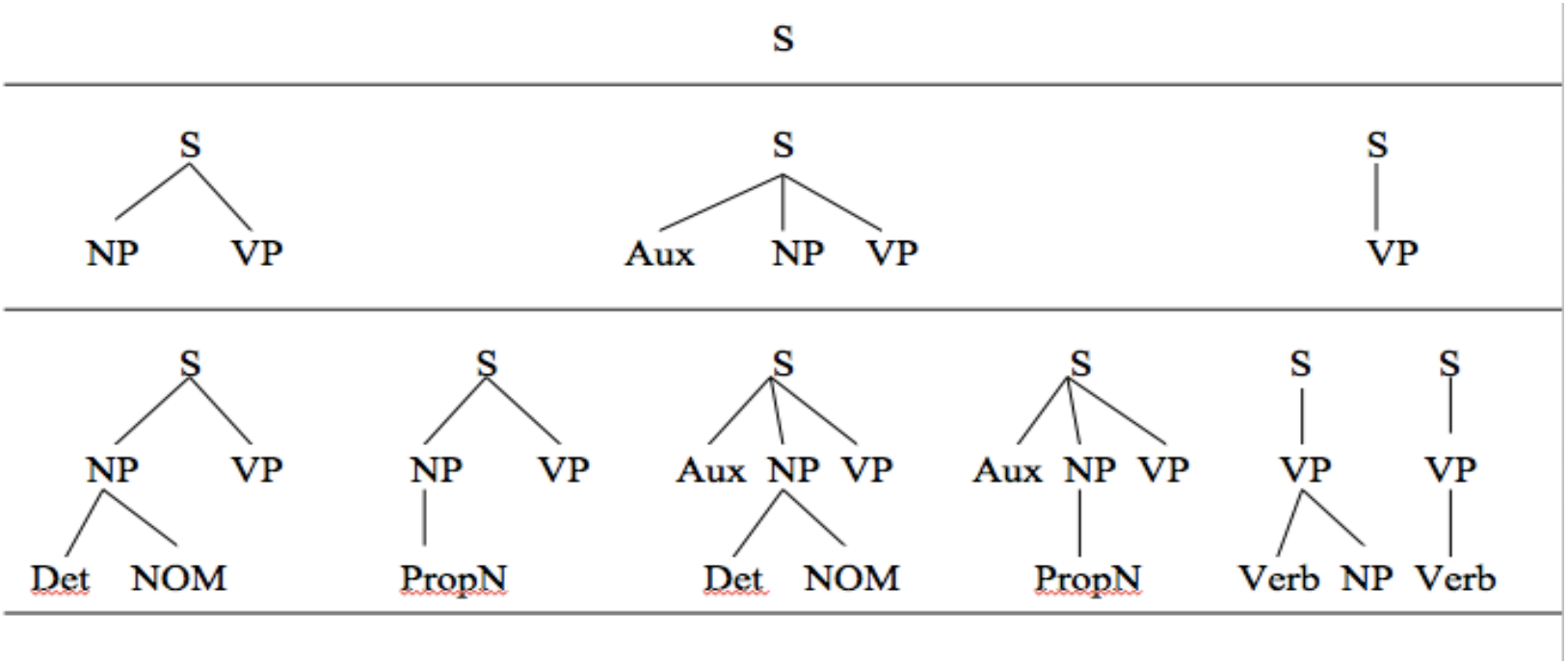
PP → Prep NOM

Det → that | this | a | the
Noun → book | flight | meal | money
Verb → book | include | prefer

Aux → does
Prep → from | to | on

ProperNoun → Houston | Washington

A Top-Down Search Space

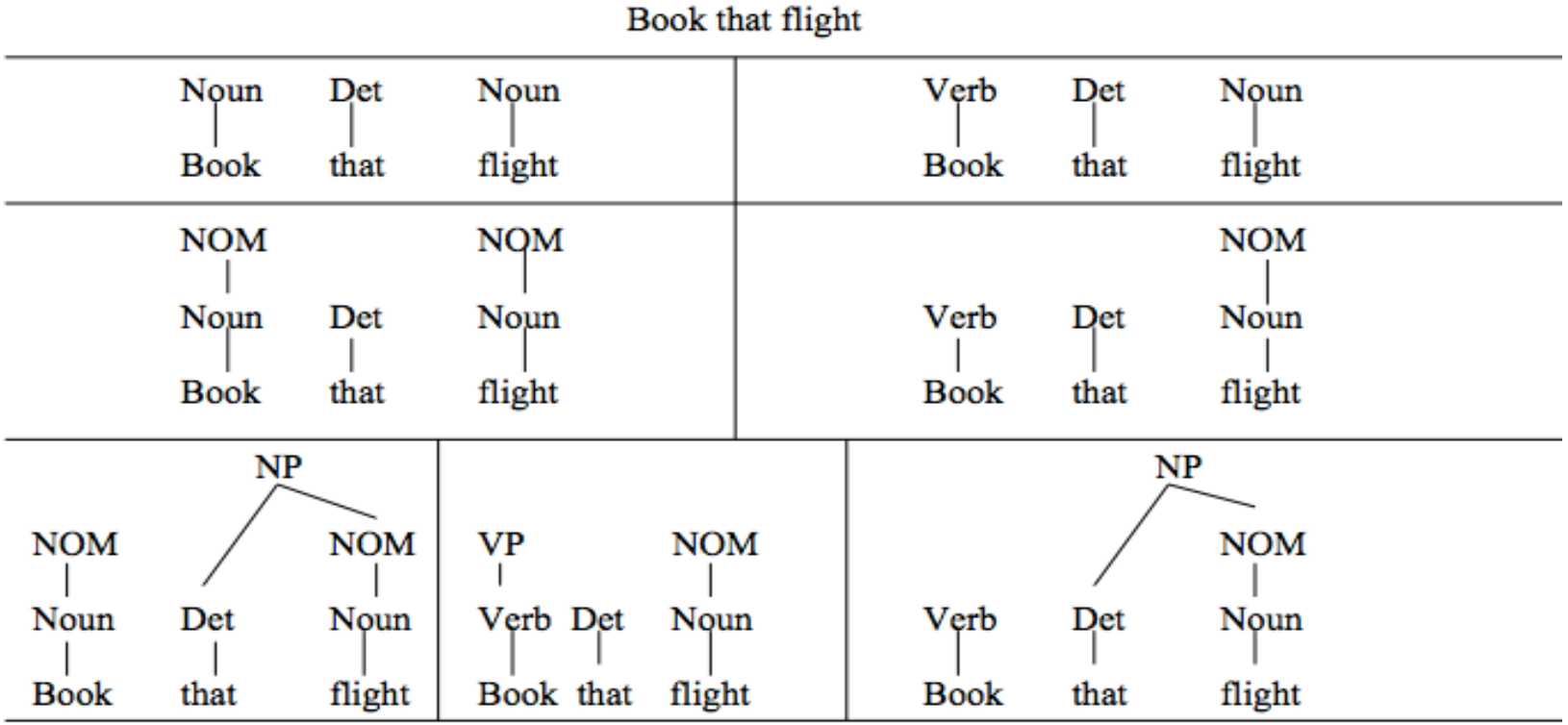


Input: Book that flight

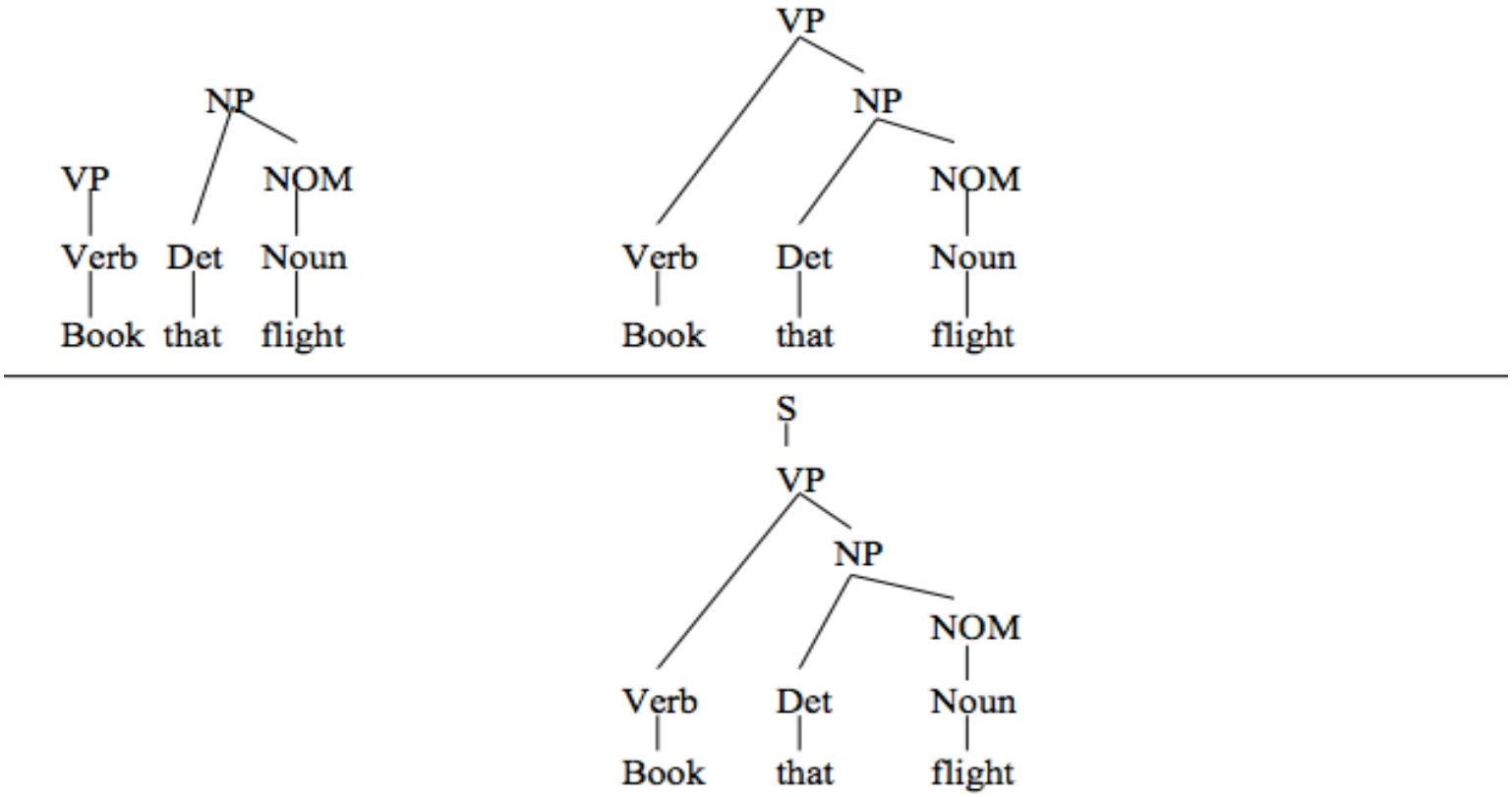
Basic Bottom-Up Parsing

- In bottom-up parsing, the parser starts with the words of input, tries to build parse trees from words up.
- The parser is successful if the parser succeeds building a parse tree rooted in the start symbol that covers all of the input.

A Bottom-Up Search Space



A Bottom-Up Search Space



Top-Down or Bottom-Up?

- Each of top-down and bottom-up parsing techniques has its own advantages and disadvantages.
- The top-down strategy never wastes time exploring trees cannot result in the start symbol (starts from there).
- On the other hand, bottom-up strategy may waste time in those kind of trees.
- But the top-down strategy spends with trees which are not consistent with the input.
- On the other hand, bottom-up strategy never suggests trees that are not at least locally grounded in the actual input.
- None of these two basic strategies are good enough to be used in the parsing of natural languages.

Ambiguity

- Top-down parser is not efficient at handling ambiguity.
- Global ambiguity potentially leads to multiple parses for the same input (if we force it to do).
- The parsers without disambiguation tools must simply return all possible parses. But most of disambiguation tools require statistical and semantic knowledge.
- There will be many unreasonable parses. But most of applications do not want all possible parses, they want a single correct parse.

An example on the board…

The CYK Algorithm

- *The membership problem*:
  - Problem:
    - Given a context-free grammar **G** and a string **w**
      - **G** = (V, ∑ ,P , S) where
        - V finite set of non-terminals (variables)
        - ∑ (the alphabet) finite set of terminal symbols
        - P finite set of rules
        - S start symbol (distinguished element of V)
        - V and ∑ are assumed to be disjoint
      - **G** is used to generate the string of a language
  - Question:
    - Is **w** in **L**(**G**)?

The CYK Algorithm

- J. Cocke
- D. Younger,
- T. Kasami

  - Independently developed an algorithm to answer this question.
  - It employs bottom-up parsing and dynamic programming.

The CYK Algorithm Basics

- The Structure of the rules in a Chomsky Normal Form grammar
- Uses a “dynamic programming” or “table-filling algorithm”

Chomsky Normal Form

- *Normal Form* is described by a set of conditions that each rule in the grammar must satisfy
- Context-free grammar is in CNF if each rule has one of the following forms:
  - $A \rightarrow BC$ at most 2 symbols on right side
  - $A \rightarrow a$ terminal symbol
  - $S \rightarrow \lambda$ null string

  where $B, C \in V - \{S\}$

Construct a Triangular Table

- Each row corresponds to one length of substrings
  - Bottom Row – Strings of length 1
  - Second from Bottom Row – Strings of length 2

    .

    .
  - Top Row – string ‘w’

Construct a Triangular Table

He was watching TV yesterday.

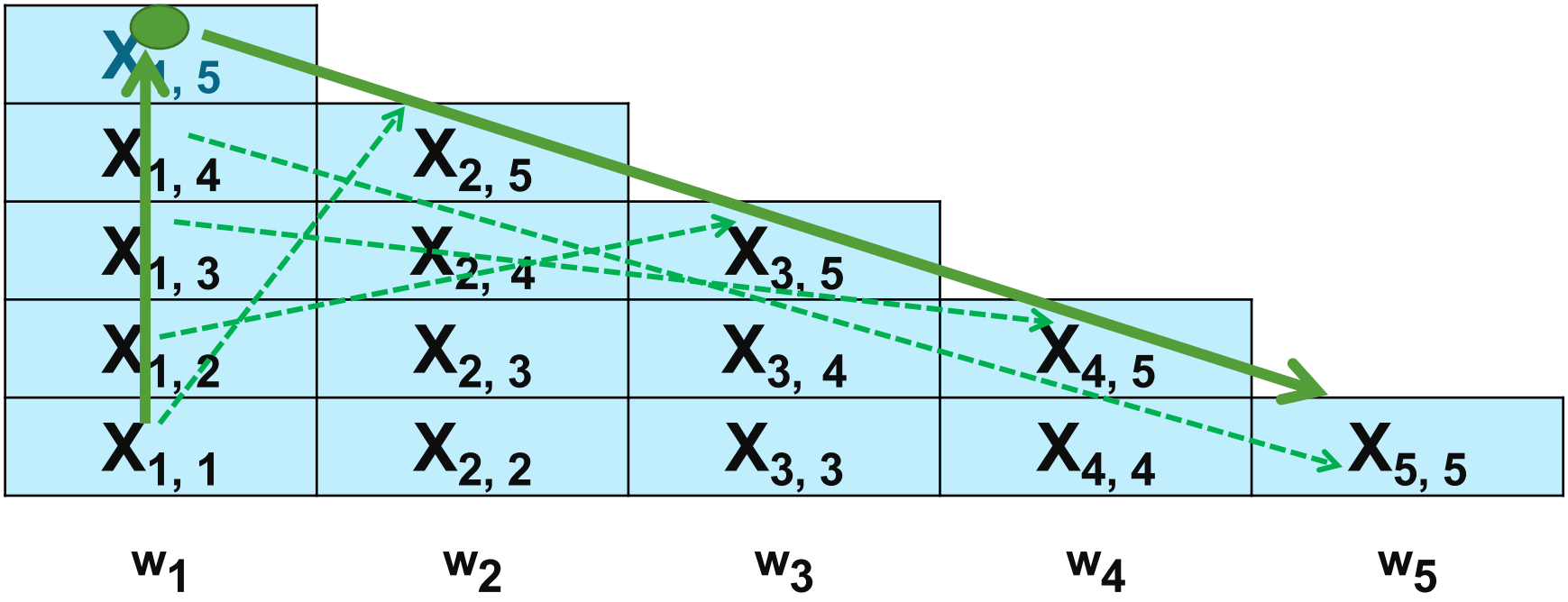
He was watching TV yesterday				
He was watching TV	was watching TV yesterday			
He was watching	was watching TV	watching TV yesterday		
He was	was watching	watching TV	TV yesterday	
He	was	watching	tv	yesterday
w_1	w_2	w_3	w_4	w_5

Construct a Triangular Table

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

Table for string '**w**' that has length 5

Construct a Triangular Table

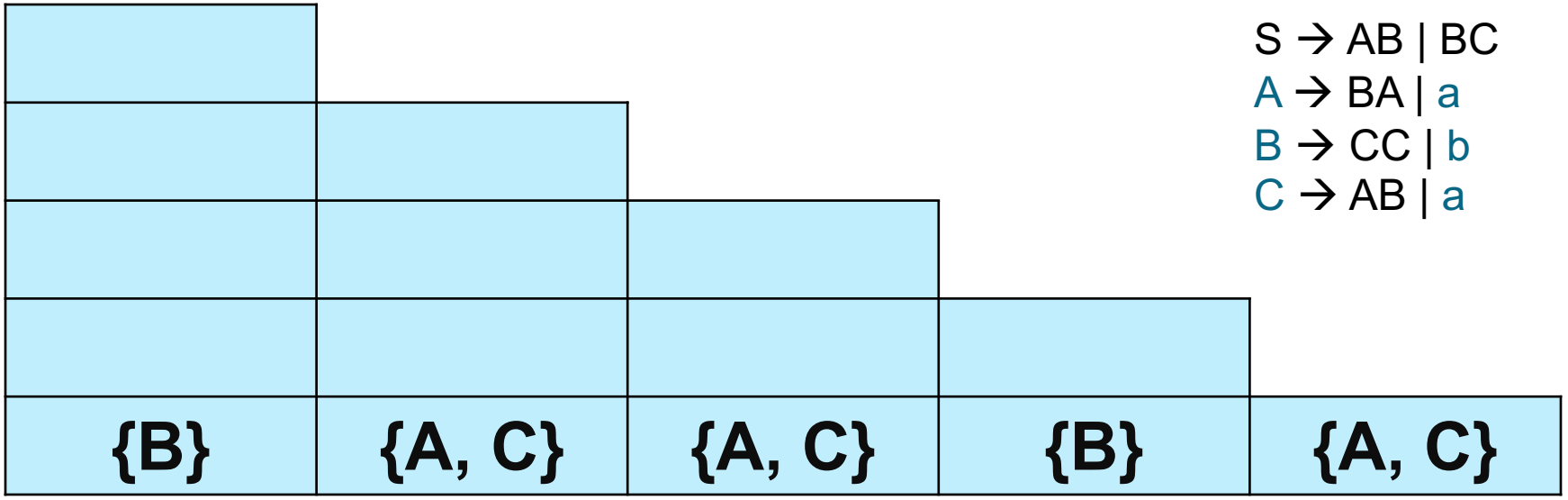


Looking for pairs to compare

Example CYK Algorithm

- 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 baaba
  - Question: Is **baaba** in L(G)?

Constructing The Triangular Table



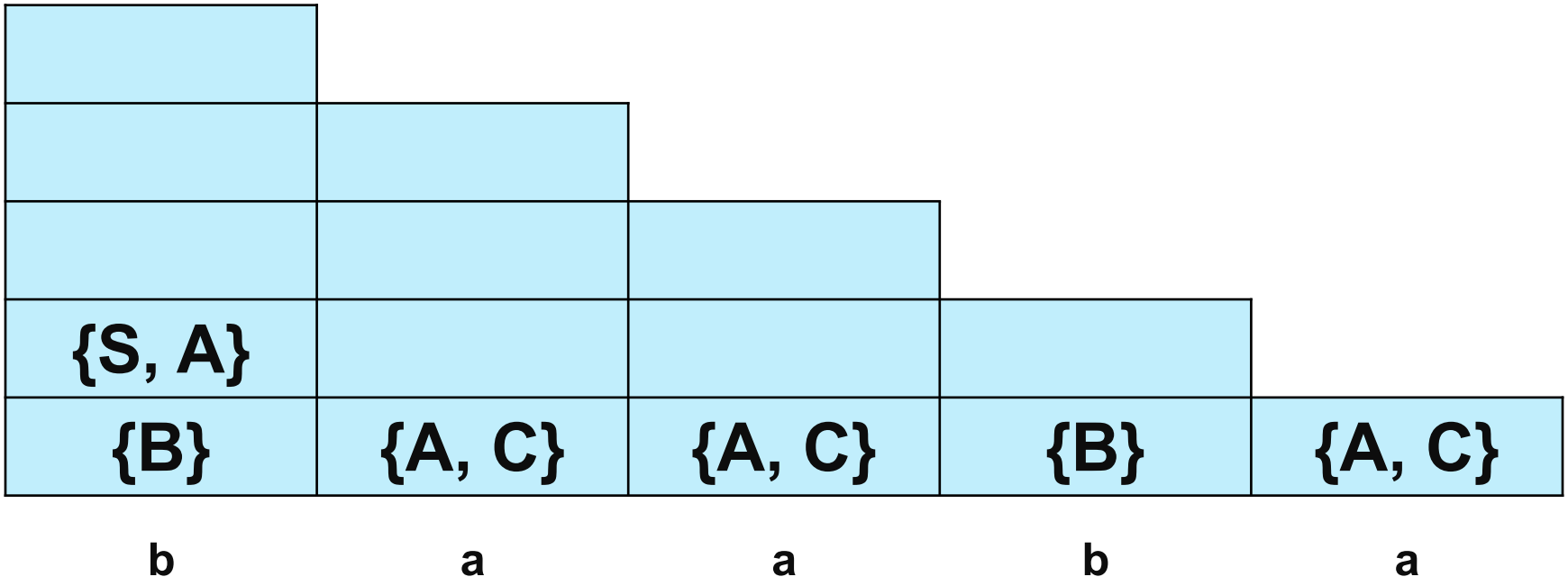
Calculating the Bottom ROW

Constructing The Triangular Table

- $X_{1,2} = (X_{i,i}, X_{i+1,j}) = (X_{1,1}, X_{2,2})$
- ➔ {B}{A,C} = {BA, BC}
- Steps:
  - Look for production rules to generate BA or BC
  - There are two: S and A
  - $X_{1,2} = \{S, A\}$

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

Constructing The Triangular Table

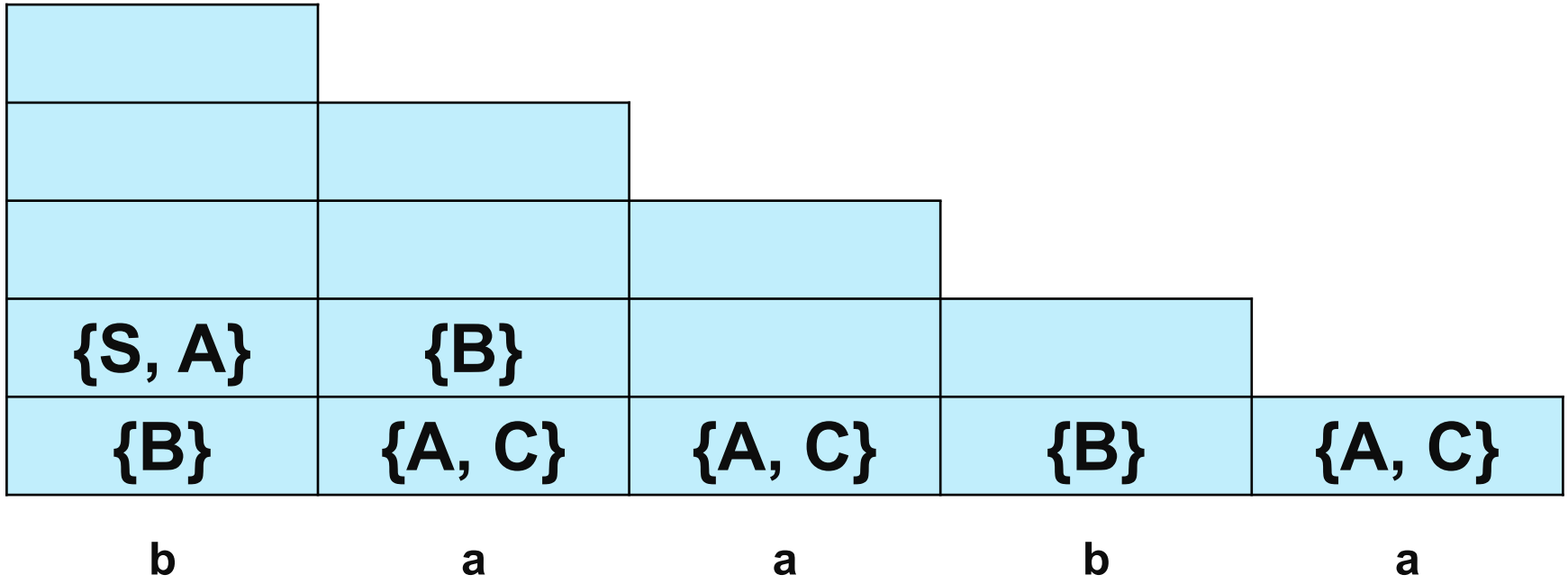


Constructing The Triangular Table

- $X_{2,3} = (X_{i,i}, X_{i+1,j}) = (X_{2,2}, X_{3,3})$
- ➔ {A, C}{A,C} = {AA, AC, CA, CC} = Y
- Steps:
  - Look for production rules to generate Y
  - There is one: B
  - $X_{2,3} = \{B\}$

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

Constructing The Triangular Table



Constructing The Triangular Table

- $X_{3,4} = (X_{i,i}, X_{i+1,j}) = (X_{3,3}, X_{4,4})$
- ➔ {A, C}{B} = {AB, CB} = Y
- Steps:
  - Look for production rules to generate Y
  - There are two: S and C
  - $X_{3,4} = \{S, C\}$

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

Constructing The Triangular Table

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

Constructing The Triangular Table

- $X_{4,5} = (X_{i,i}, X_{i+1,j}) = (X_{4,4}, X_{5,5})$
- ➔ {B}{A, C} = {BA, BC} = Y
- Steps:
  - Look for production rules to generate Y
  - There are two: S and A
  - $X_{4,5} = \{S, A\}$

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

Constructing The Triangular Table

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

Constructing The Triangular Table

- $X_{1,3} = (X_{i,i}, X_{i+1,j})\ (X_{i,i+1}, X_{i+2,j})$
  $= (X_{1,1}, X_{2,3}), (X_{1,2}, X_{3,3})$
- ➔ {B}{B} **U** {S, A}{A, C}= {BB, SA, SC, AA, AC} = Y
- Steps:
  - Look for production rules to generate Y
  - There are NONE
  - $X_{1,3} = \emptyset$
  - **no elements in this set (empty set)**

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

Constructing The Triangular Table

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

Constructing The Triangular Table

- $X_{2,4} = (X_{i,i}, X_{i+1,j})\,(X_{i,i+1}, X_{i+2,j})$
  $= (X_{2,2}, X_{3,4}), (X_{2,3}, X_{4,4})$
- ➔ {A, C}{S, C} **U** {B}{B}= {AS, AC, CS, CC, BB} = Y
- Steps:
  - Look for production rules to generate Y
  - There is one: B
  - $X_{2,4} = \{B\}$

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

Constructing The Triangular Table

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

Constructing The Triangular Table

- $X_{3,5} = (X_{i,i}, X_{i+1,j})\ (X_{i,i+1}, X_{i+2,j})$
  $= (X_{3,3}, X_{4,5}), (X_{3,4}, X_{5,5})$
- ➔ {A,C}{S,A} **U** {S,C}{A,C}

  = {AS, AA, CS, CA, SA, SC, CA, CC} = Y
- Steps:
  - Look for production rules to generate Y
  - There is one: B
  - $X_{3,5} = \{B\}$

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

Constructing The Triangular Table

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

Final Triangular Table

{S, A, C}	← $X_{1,5}$			
Ø	{S, A, C}			
Ø	{B}	{B}		
{S, A}	{B}	{S, C}	{S, A}	
{B}	{A, C}	{A, C}	{B}	{A, C}
b	a	a	b	a

- Table for string '**w**' that has length 5
- The algorithm populates the triangular table

Example (Result)

- Is baaba in L(G)?

Yes

We can see the S in the set X_{1n} where ‘n’ = 5

We can see the table

the cell X_{15} = (S, A, C) then

if S ∈ X_{15} then baaba ∈ L(G)

References

- Julia Hockenmaier, Formal Grammars of English slides
- Christopher Manning, Dan Klein, slides