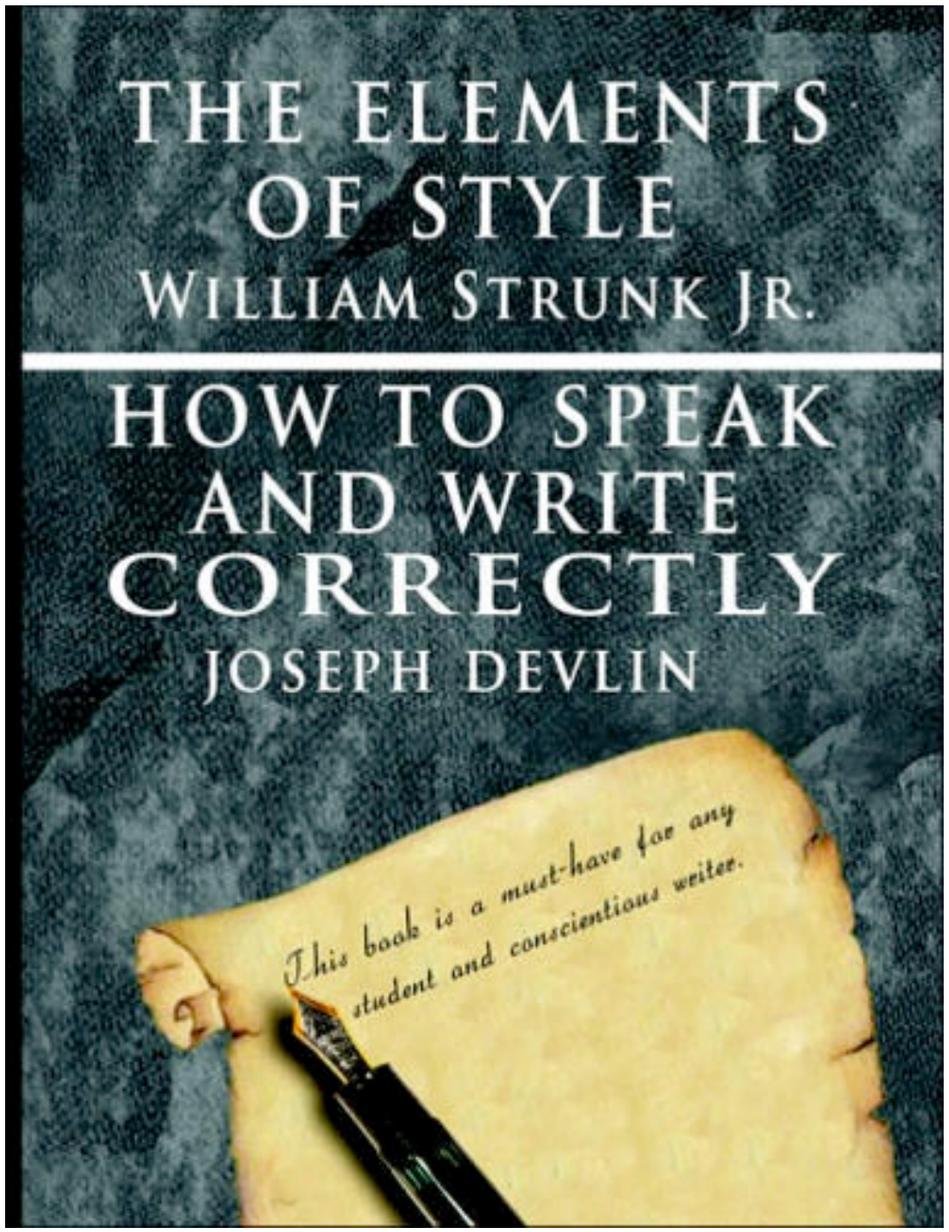
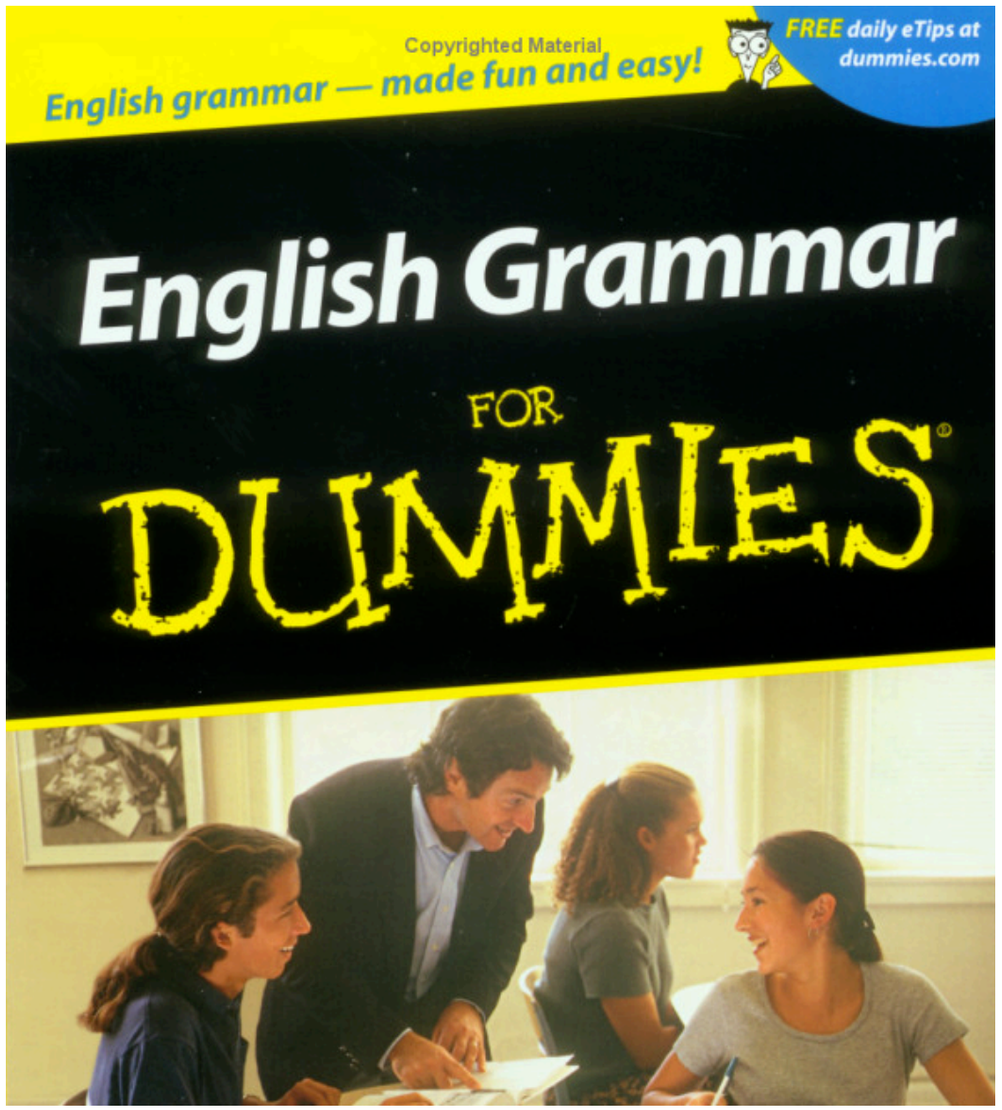
Parsing, PCFGs, and the CKY Algorithm

Yoav Goldberg

(with slides by Michael Collins, Julia Hockenmaier)

What is grammar?





Natural Language Parsing

- Sentences in natural language have structure.
- Linguists create Linguistic Theories for defining this structure.
- The parsing problem is recovering that structure.

What is grammar?

Grammar formalisms
(= linguists' programming languages)

A precise way to define and describe
the structure of sentences.

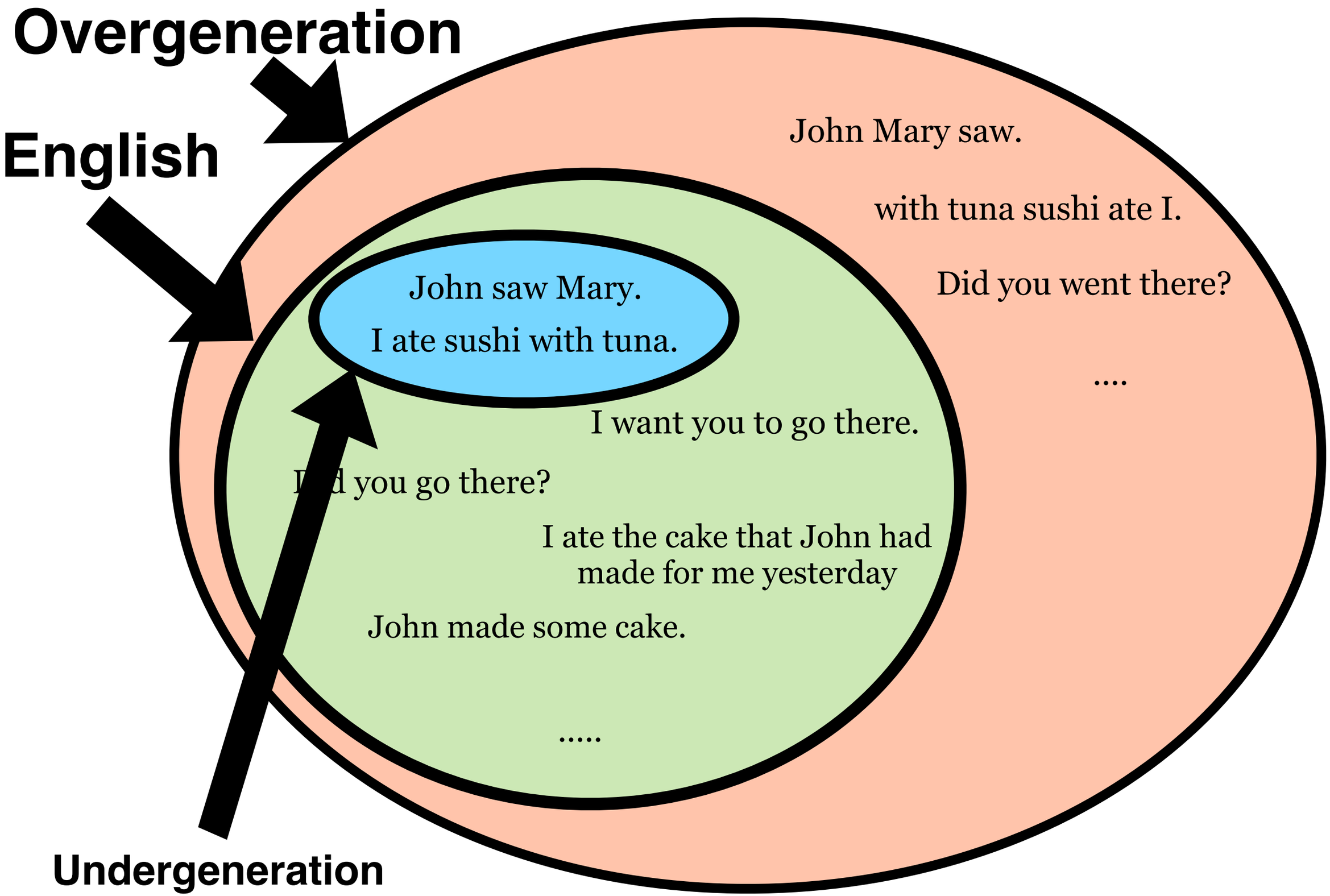
(N.B.: There are many different formalisms out there, which each define their own data structures and operations)

Specific grammars
(= linguists' programs)

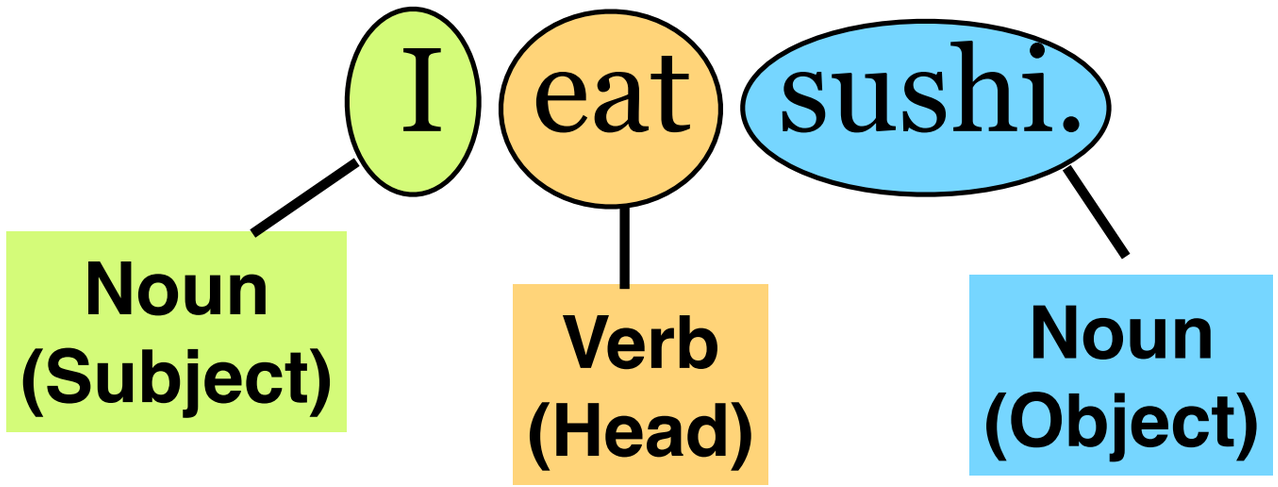
Implementations (in a particular formalism) for a particular language (English, Chinese,....)

Can we define a program that generates all English sentences?

The number of sentences is infinite.
But we need our program to be finite.

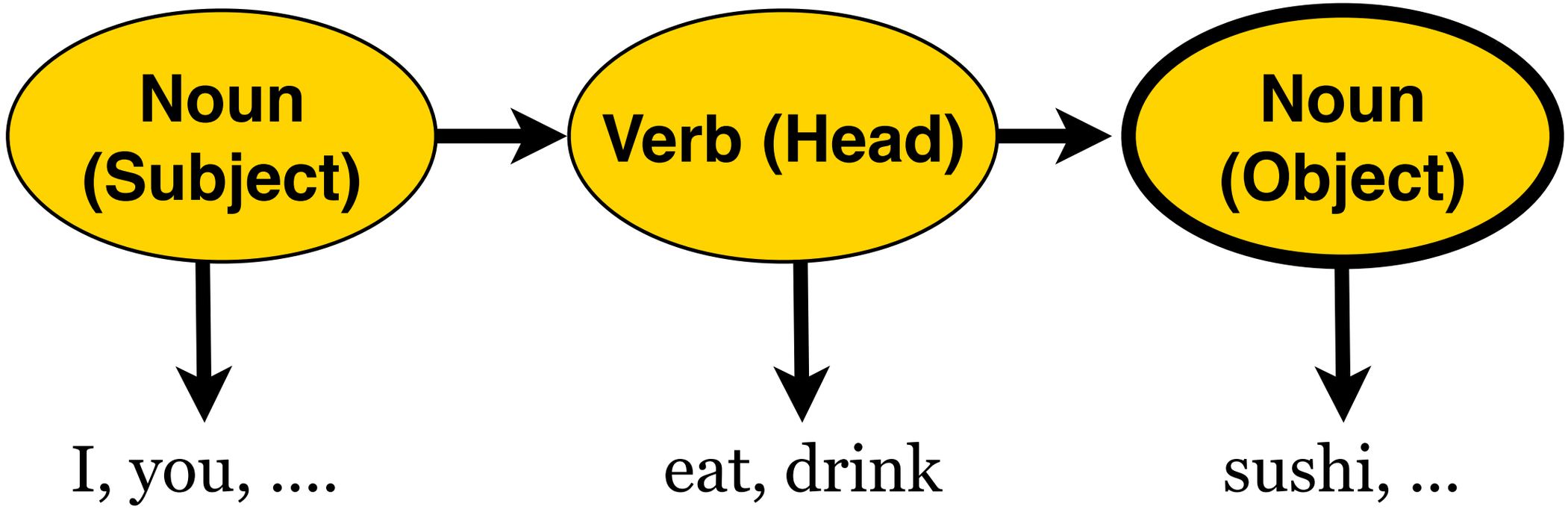

Overgeneration
English
John Mary saw.
with tuna sushi ate I.
Did you went there?
....
John saw Mary.
I ate sushi with tuna.
I want you to go there.
d you go there?
I ate the cake that John had made for me yesterday
John made some cake.
.....
Undergeneration

Basic sentence structure



A finite-state-automaton (FSA)

A Hidden Markov Model (HMM)



Words take arguments

I eat sushi. ✔
I eat sushi you. ???
I sleep sushi ???
I give sushi ???
I drink sushi ?

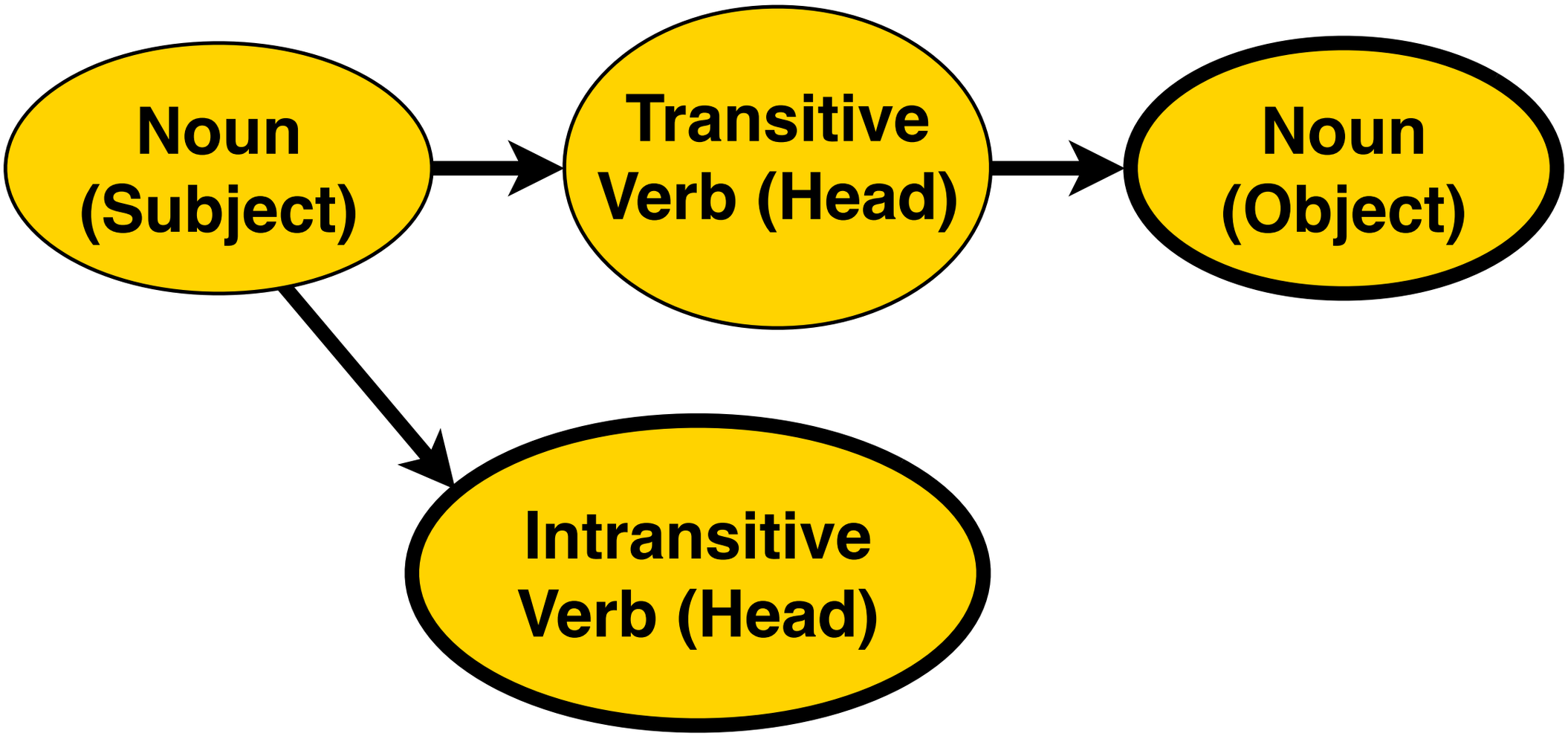
Subcategorization:

Intransitive verbs (sleep) take only a 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)

Natural Language Parsing

Previously

- Structure is a sequence.
- Each item can be tagged.
- We can mark some spans.

Natural Language Parsing

Previously

- Structure is a sequence.
- Each item can be tagged.
- We can mark some spans.

Today

- Hierarchical Structure.

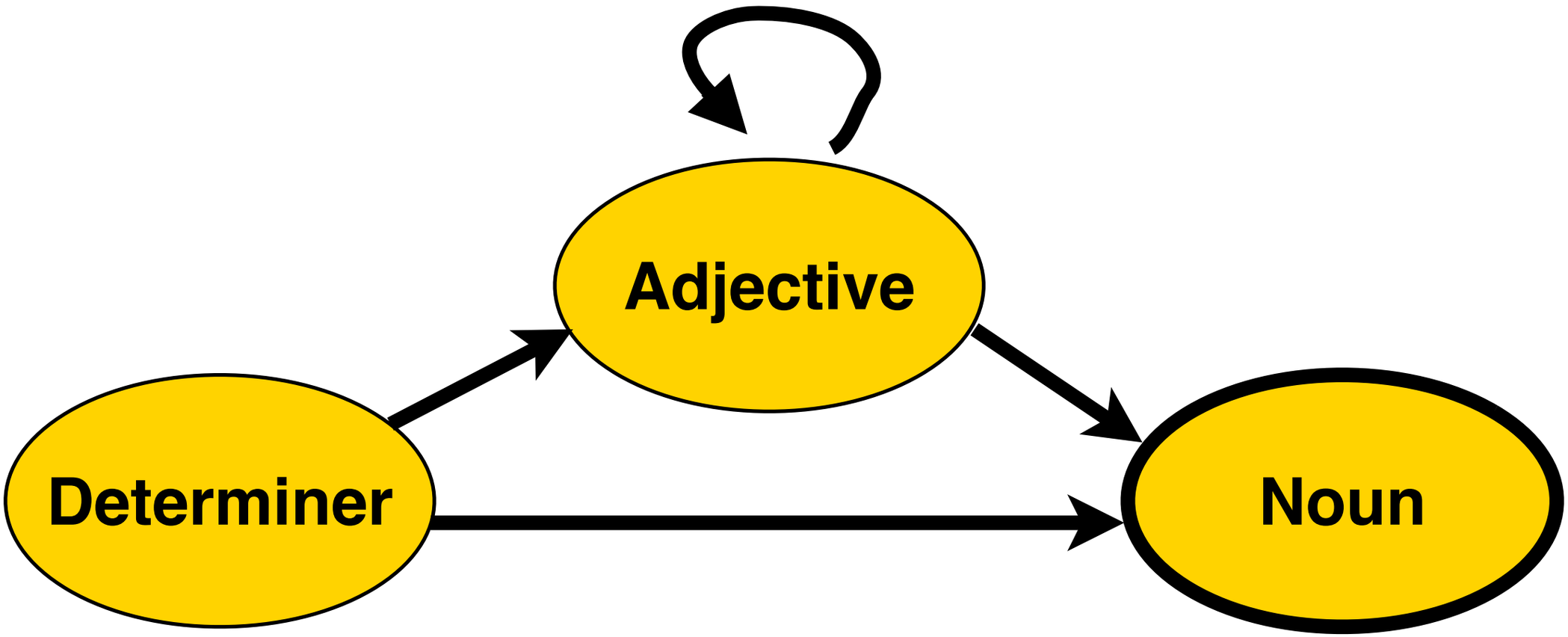
Hierarchical Structure?

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/adjuncts** a word can have is (in theory) **unlimited**.

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 the school
....

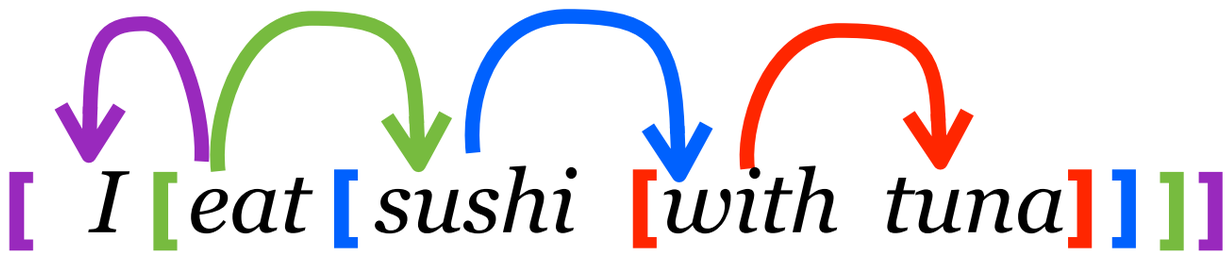
What is the structure of a sentence?

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:



Strong vs. weak generative capacity

Formal language theory:

- defines language as string sets
- is only concerned with generating these strings **(weak generative capacity)**

Formal/Theoretical syntax (in linguistics):

- defines language as sets of strings with (hidden) structure
- is also concerned with generating the right structures **(strong generative capacity)**

Structure

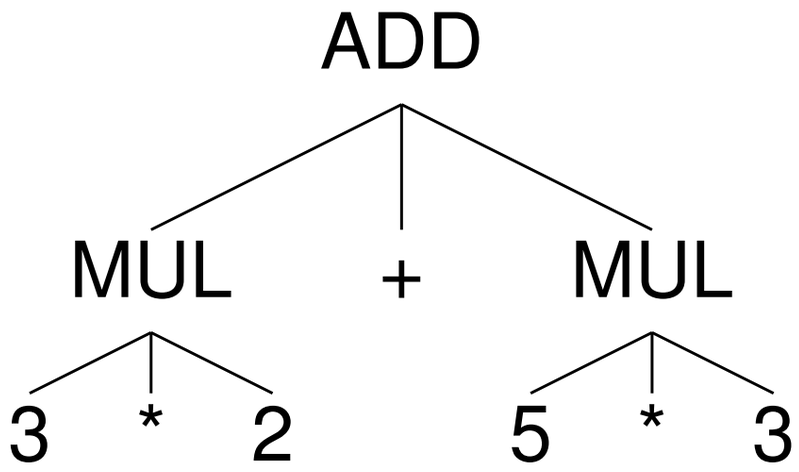
Example 1: math

3*2+5*3

Structure

Example 1: math

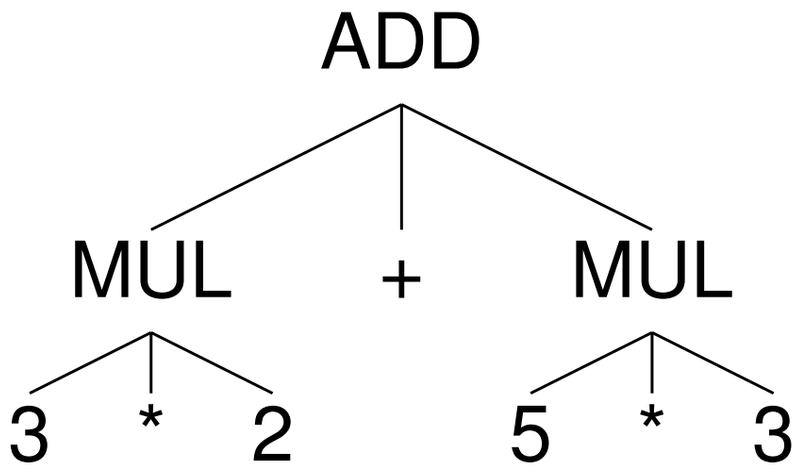
3*2+5*3

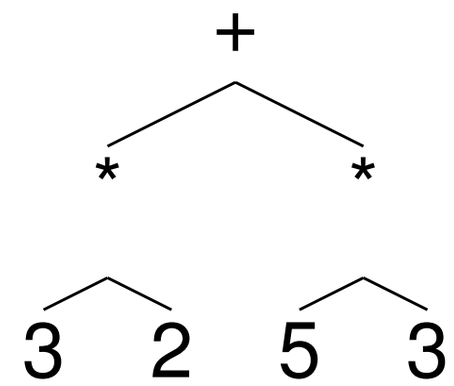


Structure

Example 1: math

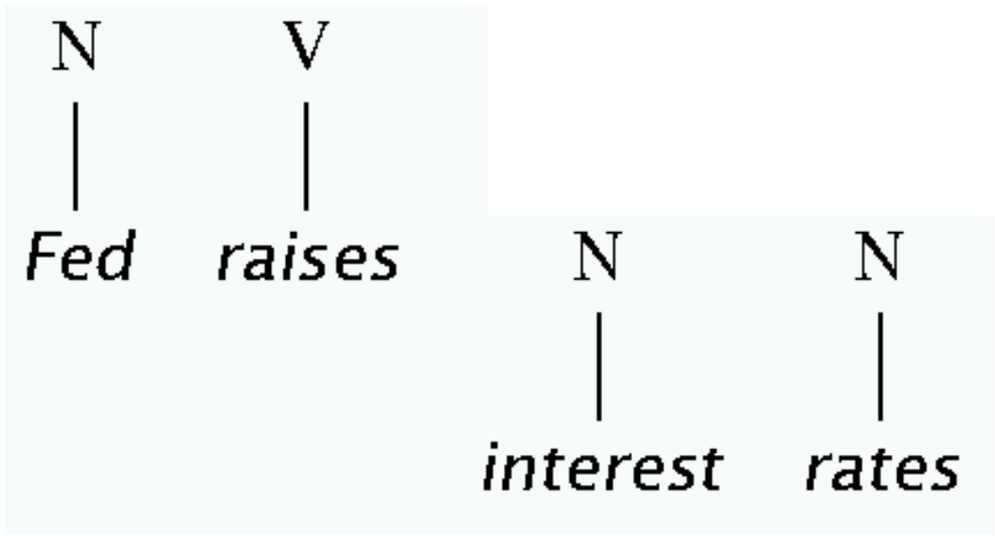
3*2+5*3





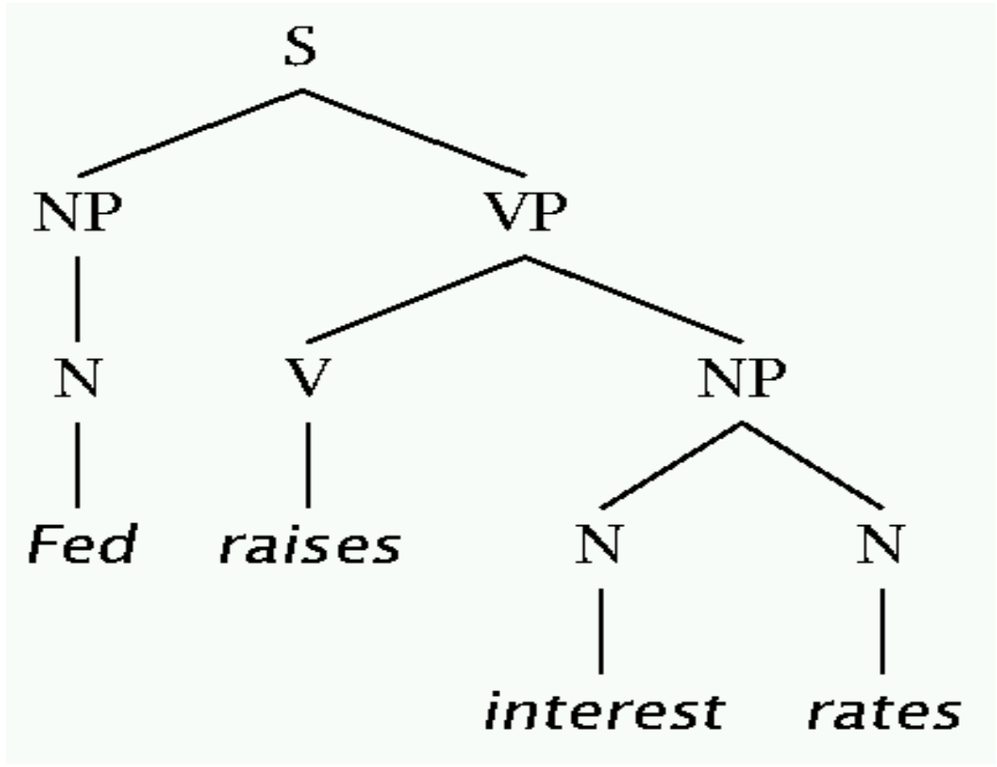
Constituency (Phrase Structure) Trees

- Phrase structure organizes words into nested constituents



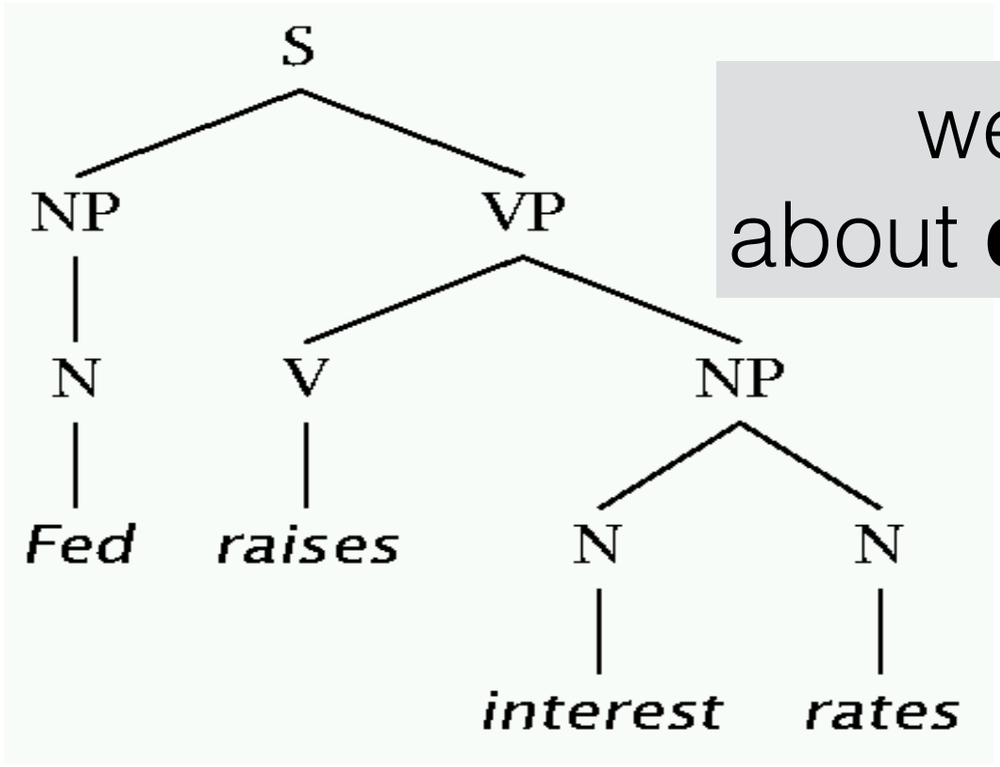
Constituency (Phrase Structure) Trees

- Phrase structure organizes words into nested constituents
- Linguists can, and do, argue about details



Constituency (Phrase Structure) Trees

- Phrase structure organizes words into nested constituents
- Linguists can, and do, argue about details



we will talk more about **constituents** soon.

Programming Languages?

They also have structure.
How does it differ from human's language structure?

Structure

Example 2: Language Data

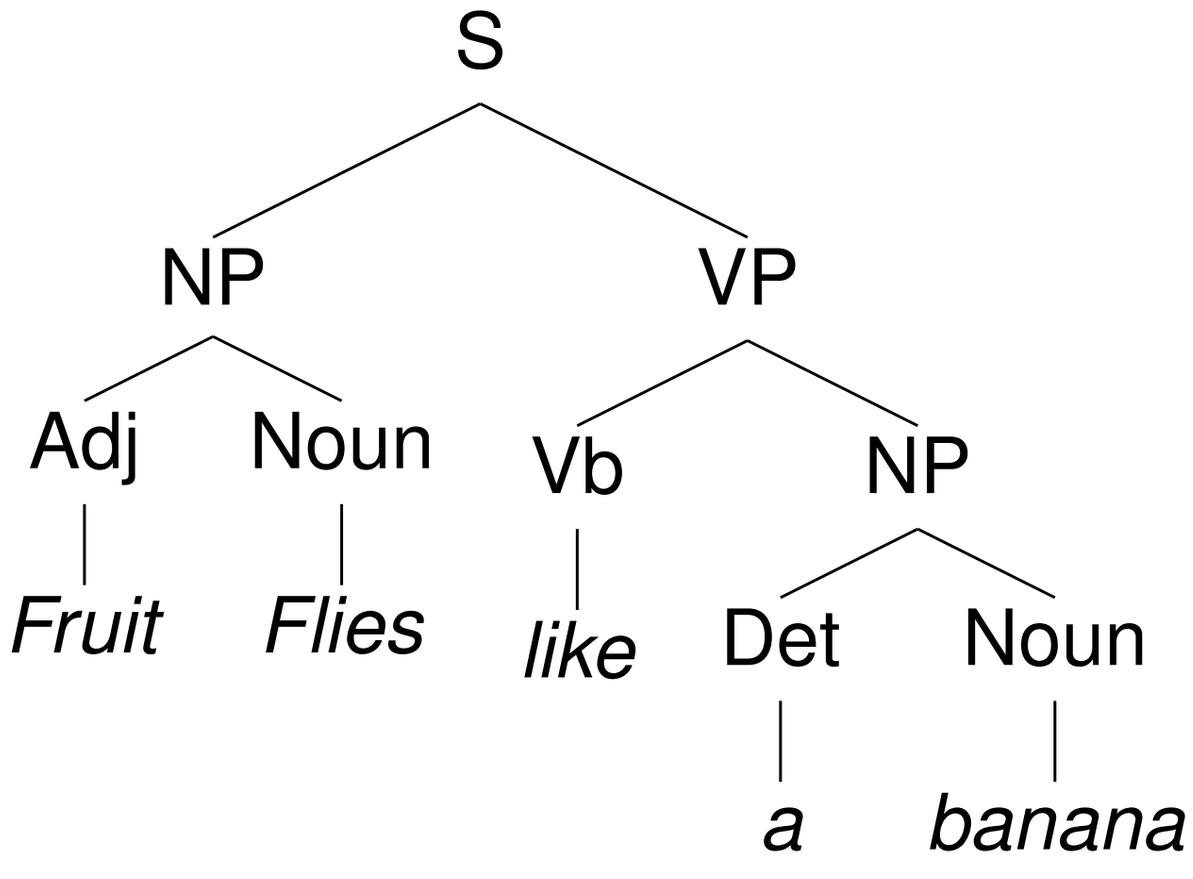
Fruit flies like a banana

Structure

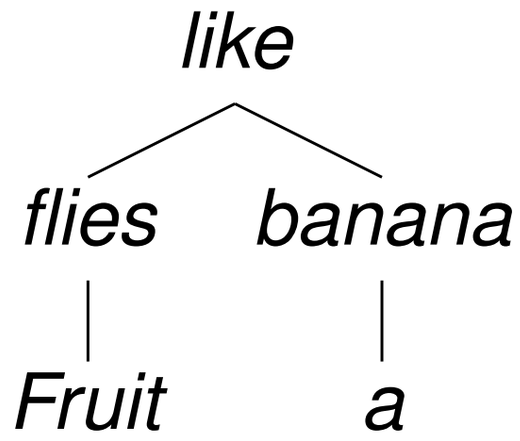
Example 2: Language Data

Fruit flies like a banana

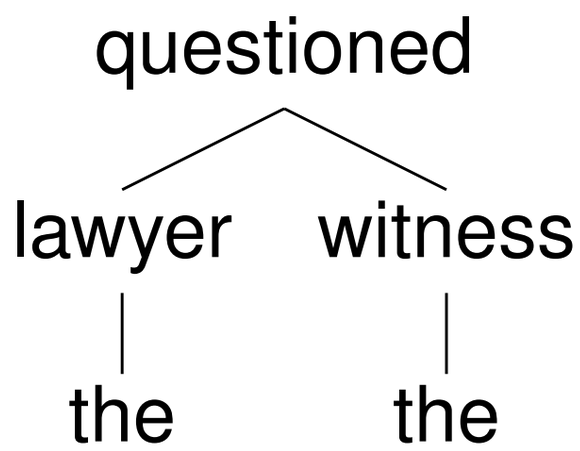
Constituency Structure

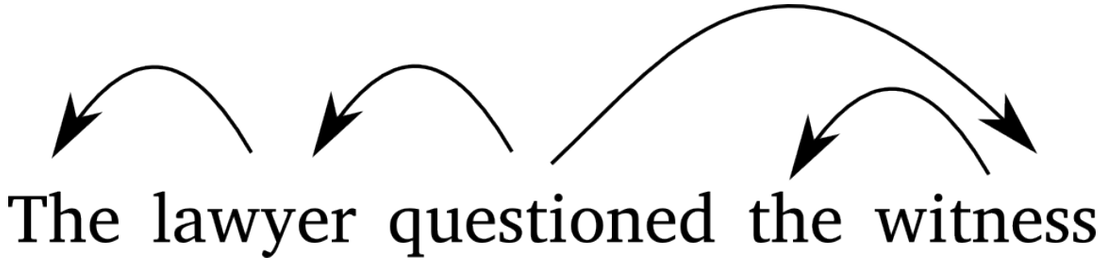


Dependency Structure

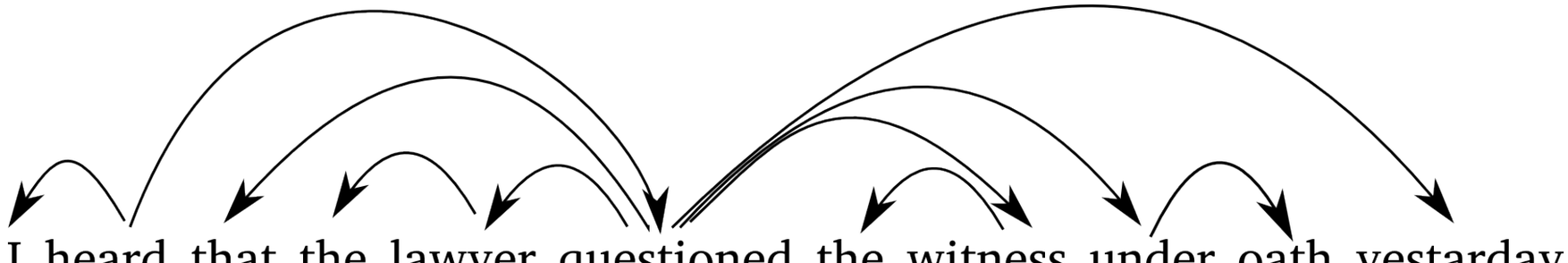


Dependency Representation

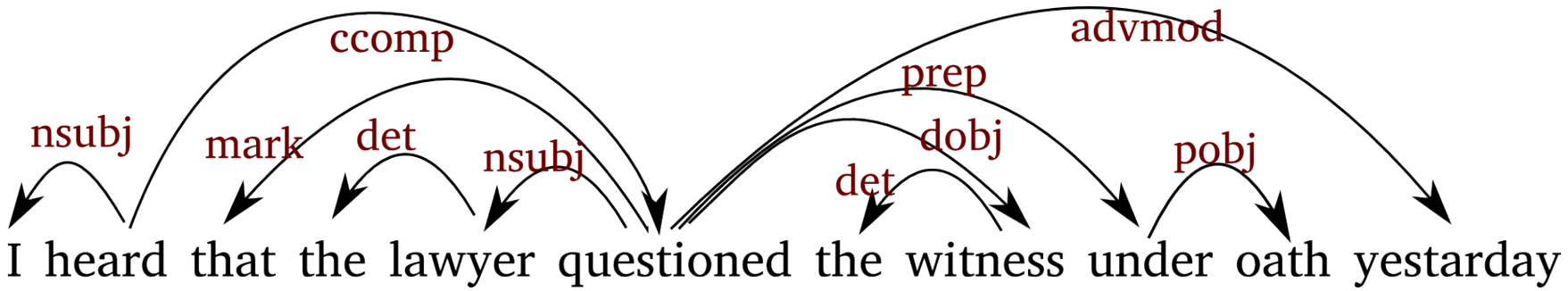




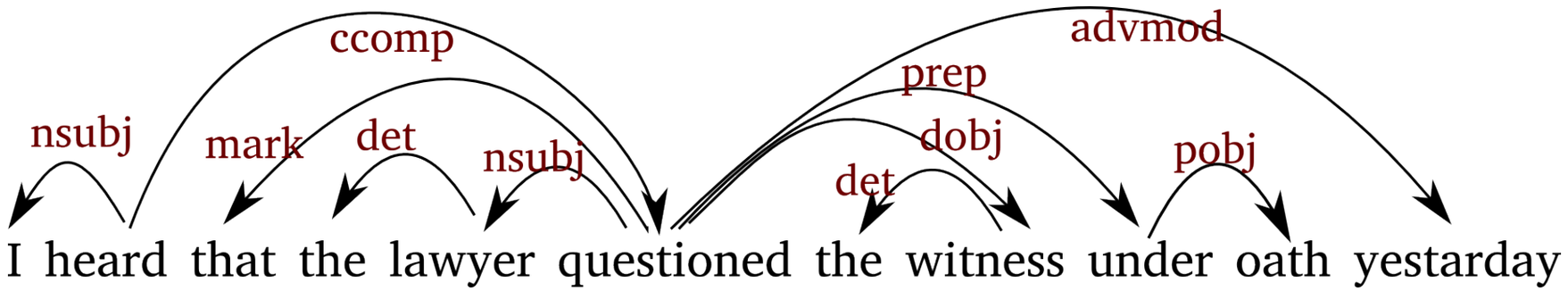
Dependency Representation



Dependency Representation



Dependency Representation



Dependency representation is very common.
We will return to it in the future.

Structure

Constituency Structure

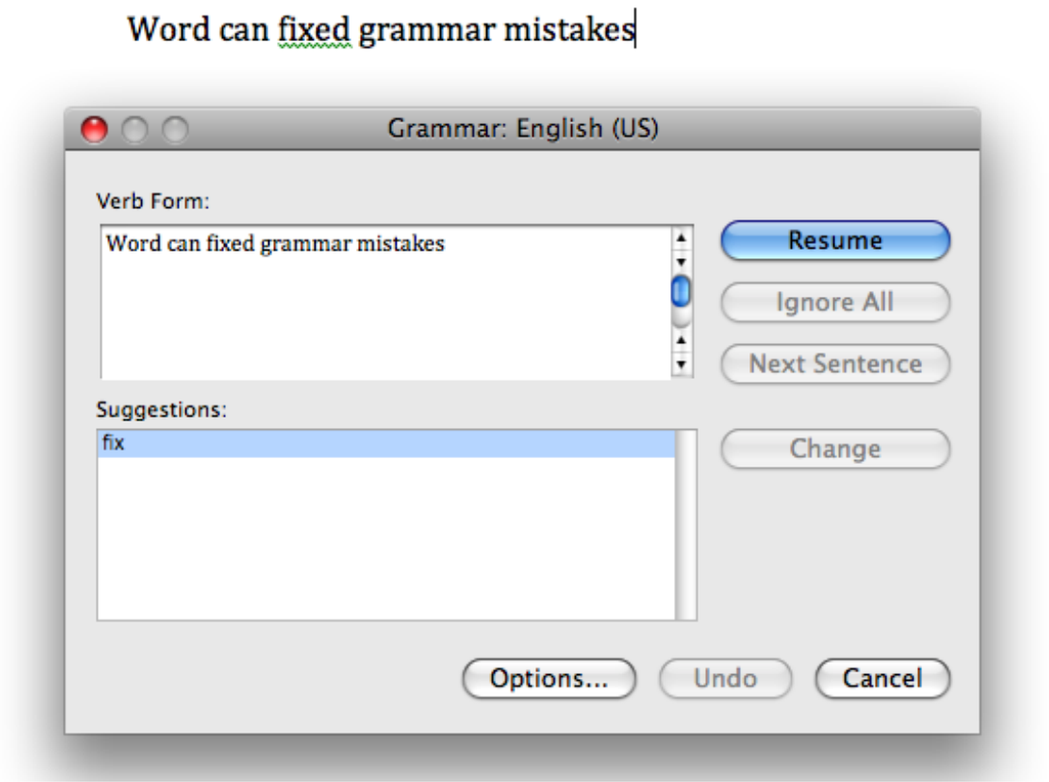
- In this class we concentrate on Constituency Parsing:
mapping from sentences to trees with labeled nodes and the sentence words at the leaves.

Why is Parsing Interesting?

- It's a first step towards understanding a text.
- Many other language tasks use sentence structure as their input.

Some things that are done with parse trees

- **Grammar Checking**
- Word Clustering
- Information Extraction
- Question Answering
- Sentence Simplification
- Machine Translation
- . . . and more



Some things that are done with parse trees

- Grammar Checking
- **Word Clustering**
- Information Extraction
- Question Answering
- Sentence Simplification
- Machine Translation
- . . . and more

Words in similar grammatical relations share meanings.

(train (desc 94476.5) (sims
recruit 0.150024
educate 0.144399
employ 0.143866
hire 0.139511
teach 0.128722
prepare 0.125677
involve 0.124054

(train (desc 129401) (sims
bus 0.255581
"passenger train" 0.199742
"freight train" 0.182971
truck 0.175791
"express train" 0.163199
plane 0.160376
ferry 0.153743
car 0.152089
boat 0.149375

(vodka (desc 16439.9) (sims
rum 0.234184
whiskey 0.22456
tequila 0.223834
Cognac 0.221328
whisky 0.210289
bourbon 0.203362
liqueur 0.185826

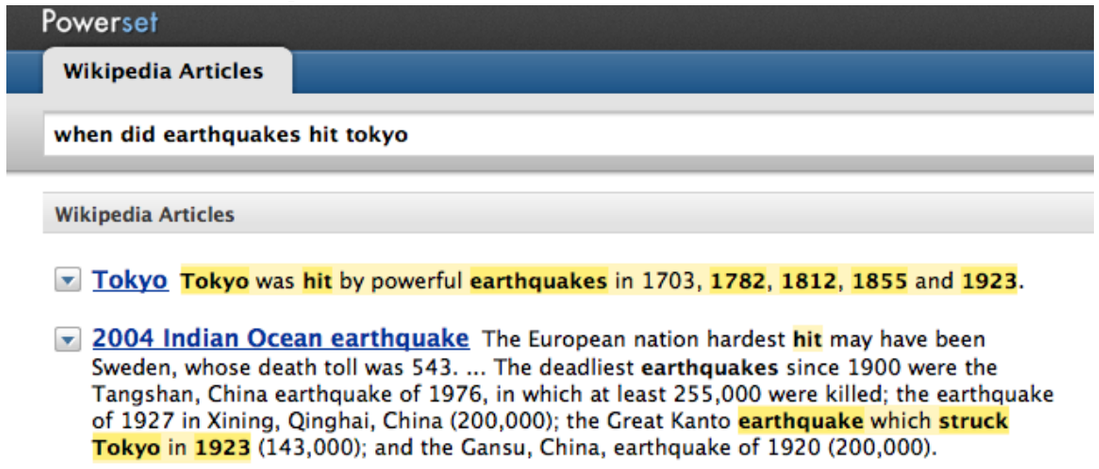
(pizza (desc 38638) (sims
sandwich 0.217842
salad 0.178421
"hot dog" 0.176298
Hamburger 0.175821
pasta 0.169979
steak 0.167151
soup 0.165197
bread 0.164151
"french fries" 0.164104
chicken 0.158617
pie 0.158087

Some things that are done with parse trees

- Grammar Checking
- Word Clustering
- **Information Extraction**
- **Question Answering**
- Sentence Simplification
- Machine Translation
- . . . and more

Extract factual relations from text

Answer questions



Some things that are done with parse trees

- Grammar Checking
- Word Clustering
- Information Extraction
- Question Answering
- **Sentence Simplification**
- Machine Translation
- . . . and more

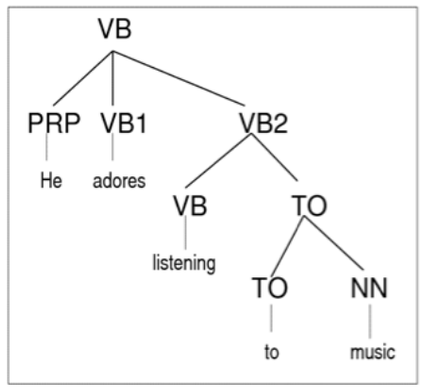
The first new product , ATF Protype , is a line of digital postscript typefaces that will be sold in packages of up to six fonts .

ATF Protype is a line of digital postscript typefaces will be sold in packages of up to six fonts.

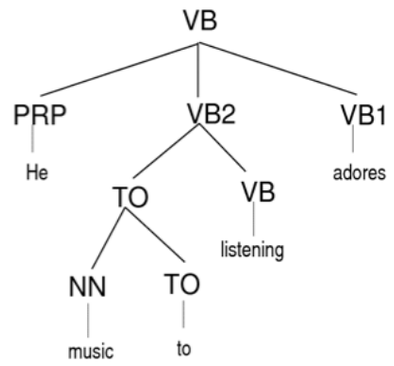
Some things that are done with parse trees

- Grammar Checking
- Word Clustering
- Information Extraction
- Question Answering
- Sentence Simplification
- **Machine Translation**
- . . . and more

Reorder

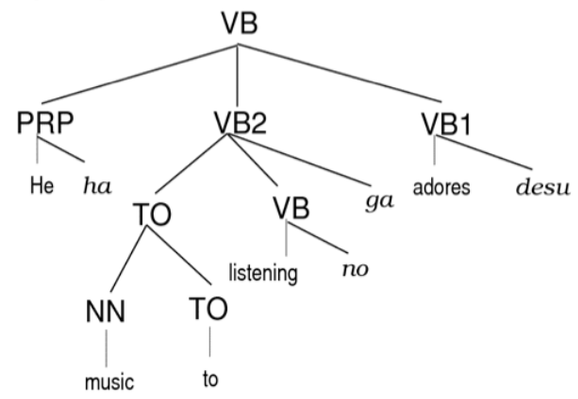


1. Original Parse Tree

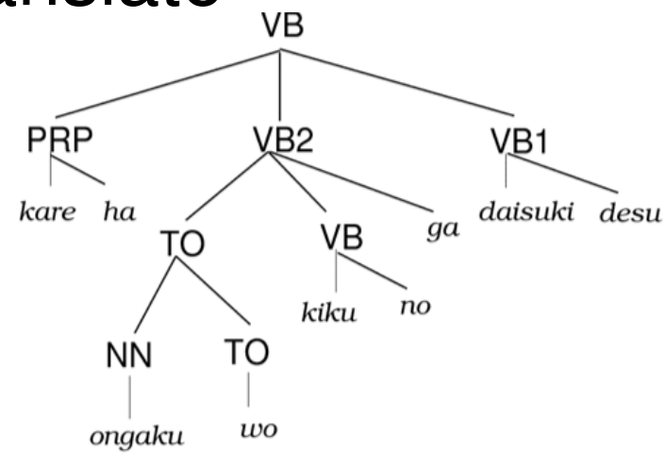


2. After Reorder

Insert



Translate



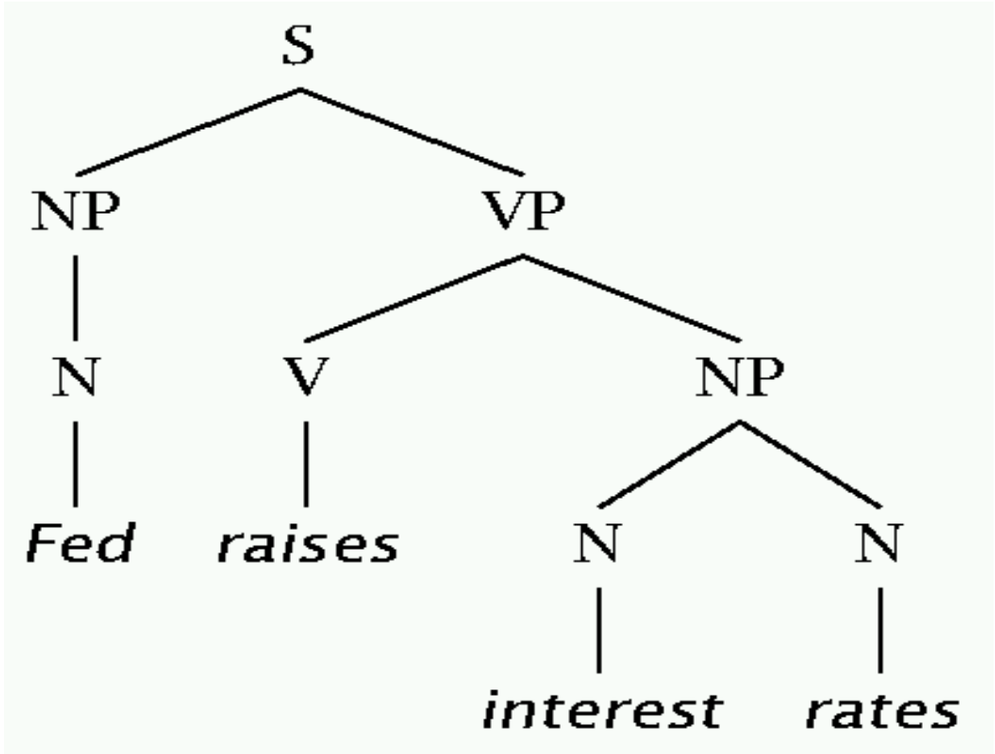
Some things that are done with parse trees

- Grammar Checking
- Word Clustering
- Information Extraction
- Question Answering
- Sentence Simplification
- Machine Translation
- **. . . and more**

Constituency (Phrase Structure) Trees

- Phrase structure organizes words into nested constituents

What is a constituent?
How do we know they exist?



Context-free grammars (CFGs) capture recursion

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

Syntactically, these constituents behave just like simple ones.
(“behind the house” can always be omitted)

CFGs define nonterminal categories to capture equivalent constituents.

The black dog sat

is "dog" a constituent?

is "The black dog" a constituent?

is "the black" a constituent?

is "dog sat" a constituent?

why?

Substitution Test

Every word is a constituent.

If we can substitute a sequence of words by a single word, it is likely to be a constituent.

In particular, pronouns.

(based on book chapter by Beatrice Santorini)

Substitution Test

(1) a. The little boy fed the cat. → He fed her.

b. Black cats detest green peas. → They detest them.

(based on book chapter by Beatrice Santorini)

Substitution Test

(2) a. The little boy from next door fed the cat without a tail.

b. These black cats detest those green peas.

(based on book chapter by Beatrice Santorini)

Substitution Test

(2) a. The little boy from next door fed the cat without a tail.

b. These black cats detest those green peas.

→ * He from next door fed her without a tail.

→ * These they detest those them.

(based on book chapter by Beatrice Santorini)

Substitution Test

(3) a. The little boy from next door fed the cat without a tail.

b. These black cats detest those green peas.

→ He fed her.

→ They detest them.

(based on book chapter by Beatrice Santorini)

Substitution Test

adverbs (here/there) can also be used.

(4) a. Put it <u>on the table</u>. → Put it <u>there</u>.
b. Put it over <u>on the table</u>. → Put it over <u>there</u>.
c. Put it over on the table. → Put it there.

(5) a. Put it <u>on the table</u> that's by the door. → * Put it <u>there</u> that's by the door.
b. Put it over <u>on the table</u> that's by the door. → * Put it over <u>there</u> that's by the door.
c. Put it <u>over on the table</u> that's by the door. → * Put it <u>there</u> that's by the door.

(based on book chapter by Beatrice Santorini)

Substitution Test

"so",
In a comparison.

(6) a. I am very happy, and Linda is so, too.
b. I am very fond of Lukas, and Linda is so, too.
c. I am very fond of my nephew, * and Linda is so of her niece.

(based on book chapter by Beatrice Santorini)

Substitution Test

'so', 'it', instead of a subordinate clause.

(7) a. I { know, suspect } <u>that they're invited</u>. → I { know, suspect } <u>it</u>.

b. I { imagine, think } <u>that they're invited</u>. → I { imagine, think } <u>so</u>.

(based on book chapter by Beatrice Santorini)

Substitution Test

Substitution tests are a good indicator of constituency.

But they may fail, also for real constituents.

There are other tests: **movement, short answers, it clefts**

We will briefly cover movement.
See more in the Santorini's chapter.

(based on book chapter by Beatrice Santorini)

Movement Test

Can we move the sequence
to a different position in the sentence?

(based on book chapter by Beatrice Santorini)

Movement Test

(8) a. I fed the cats.

b. I fed the cats with long, fluffy tails.

→ The cats, I fed ___. (The dogs, I didn't.)

→ The cats with long, fluffy tails, I fed ___. (The other cats, I didn't.)

(based on book chapter by Beatrice Santorini)

Movement Test

(9) a. Prepositional phrase: The cat strolled across the porch <u>with a confident air</u>.

b. Adjective phrase: Ali Baba returned from his travels <u>wiser than before</u>.

c. Adverb phrase: They arrived at the concert hall <u>more quickly than they had expected</u>.

→ <u>With a confident air</u>, the cat strolled across the porch ___.

→ <u>Wiser than before</u>, Ali Baba returned from his travels ___.

→ <u>More quickly than they had expected</u>, they arrived at the concert hall ___.

(based on book chapter by Beatrice Santorini)

Movement Test

(10) a. I fed the cats with long, fluffy tails.

b. The cat strolled across the porch with a confident air.

c. Ali Baba returned from his travels wiser than before.

d. They arrived at the concert hall more quickly than they had expected.

(based on book chapter by Beatrice Santorini)

Movement Test

(10) a. I fed <u>the cats</u> with long, fluffy tails.

b. The cat strolled across the porch <u>with a</u> confident air.

c. Ali Baba returned from his travels <u>wiser than</u> before.

d. They arrived at the concert hall <u>more quickly than</u> <u>they</u> had expected.

→ * <u>The cats</u>, I fed ___ with long, fluffy tails.[1]

→ * <u>With a</u>, the cat strolled across the porch ___ confident air.

→ * <u>Wiser than</u>, Ali Baba returned from his travels ___ before.

→ * <u>More quickly than they</u>, they arrived at the concert hall ___ had expected.

(based on book chapter by Beatrice Santorini)

See more tests in the notes.

I hope that you are convinced that constituents are "real".

You need to know how to identify them.

Parsing: recovering the constituents of a sentence.

Why is parsing hard?

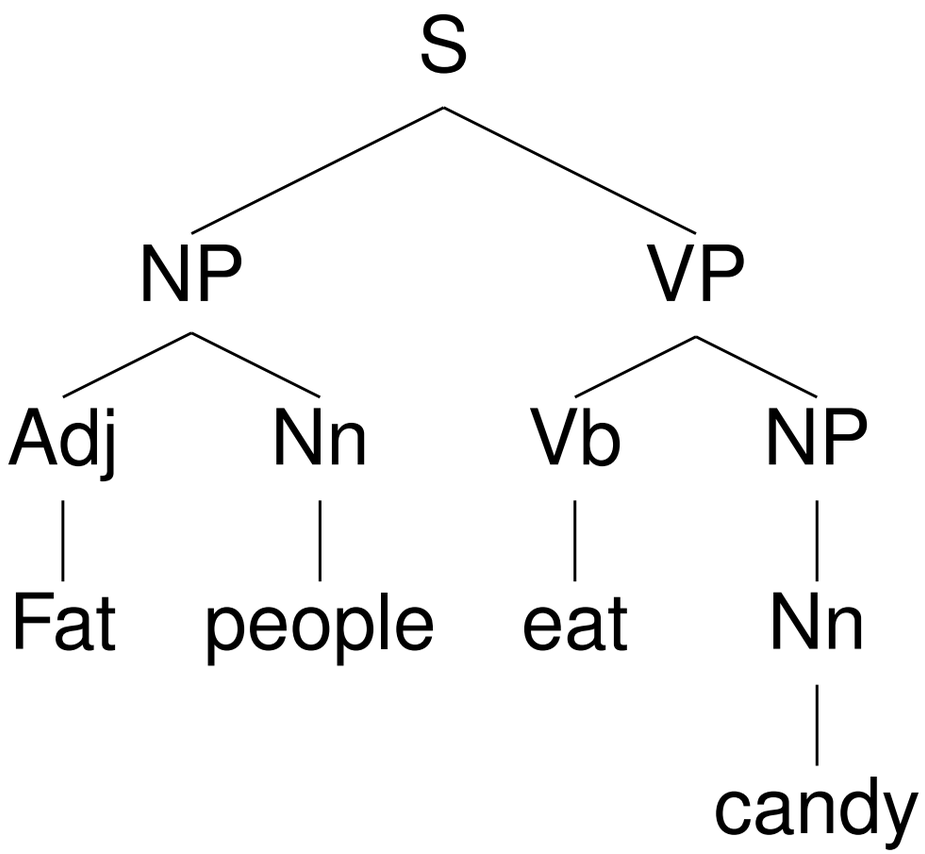
Ambiguity

Fat people eat candy

Why is parsing hard?

Ambiguity

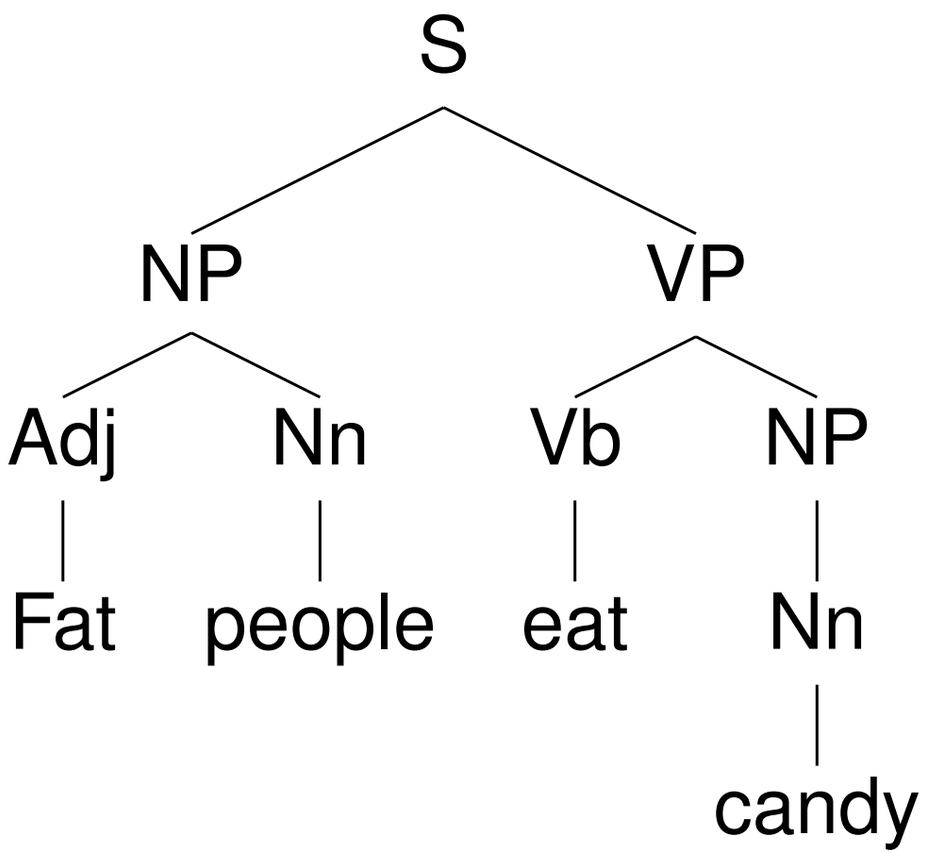
Fat people eat candy



Why is parsing hard?

Ambiguity

Fat people eat candy

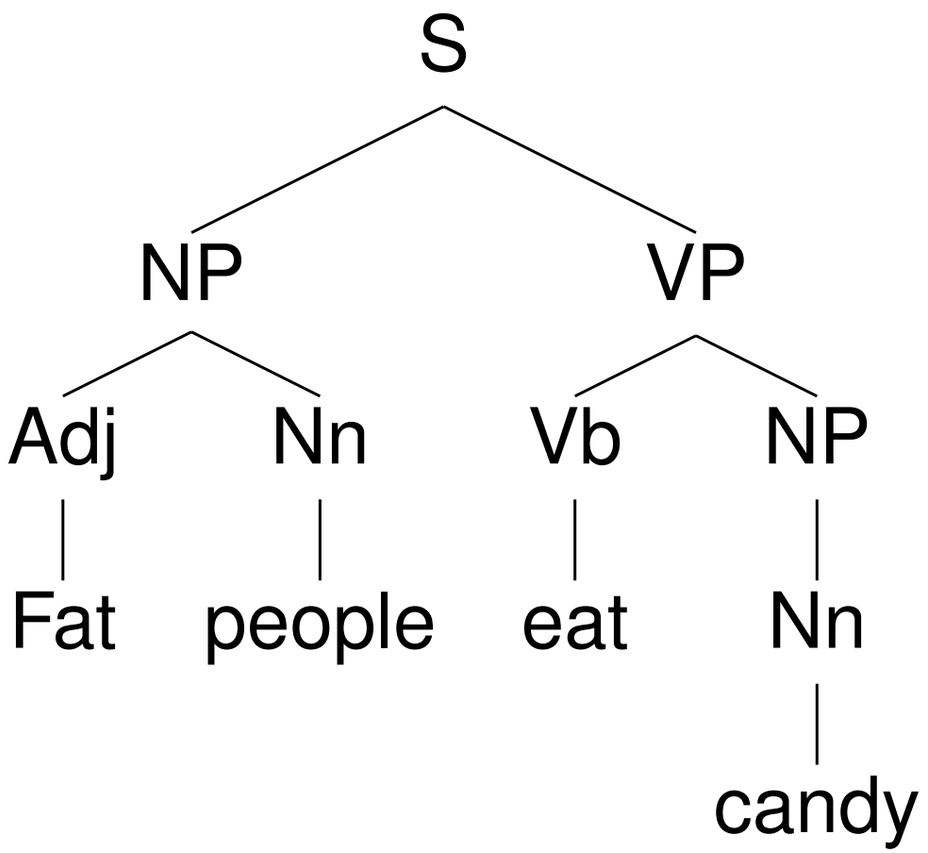


Fat people eat accumulates

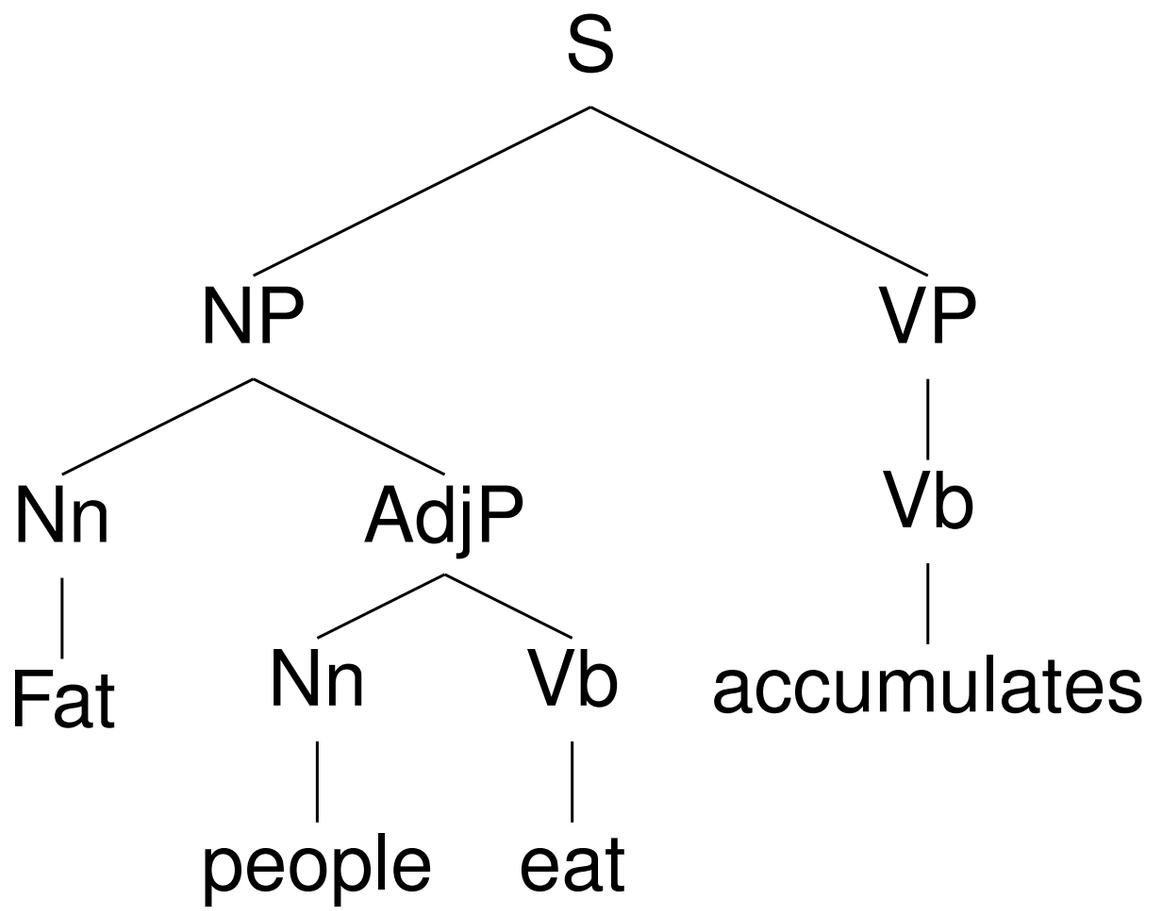
Why is parsing hard?

Ambiguity

Fat people eat candy



Fat people eat accumulates



Why is parsing hard?

Real Sentences are long. . .

“Former Beatle Paul McCartney today was ordered to pay nearly $50M to his estranged wife as their bitter divorce battle came to an end . ”

“Welcome to our Columbus hotels guide, where you’ll find honest, concise hotel reviews, all discounts, a lowest rate guarantee, and no booking fees.”

Let’s learn how to parse

Let’s learn how to parse . . . but first lets review some stuff we learned at Automata and Formal Languages.

Context Free Grammars

A context free grammar $G = (N, \Sigma, R, S)$ where:

- N is a set of non-terminal symbols
- Σ is a set of terminal symbols
- R is a set of rules of the form $X \to Y_1 Y_2 \cdots Y_n$ for $n \geq 0$, $X \in N$, $Y_i \in (N \cup \Sigma)$
- $S \in N$ is a special start symbol

Context Free Grammars

a simple grammar

$N = \{S, NP, VP, Adj, Det, Vb, Noun\}$

$\Sigma = \{fruit, flies, like, a, banana, tomato, angry\}$

$S =$‘S’

$R =$

S $\rightarrow$ NP VP

NP $\rightarrow$ Adj Noun

NP $\rightarrow$ Det Noun

VP $\rightarrow$ Vb NP

Adj $\rightarrow$ fruit

Noun $\rightarrow$ flies

Vb $\rightarrow$ like

Det $\rightarrow$ a

Noun $\rightarrow$ banana

Noun $\rightarrow$ tomato

Adj $\rightarrow$ angry

Left-most derivations

Left-most derivation is a sequence of strings $s_1, \cdots, s_n$ where

- $s_1 = S$ the start symbol
- $s_n \in \Sigma^*$, meaning s_n is only terminal symbols
- Each s_i for $i = 2 \cdots n$ is derived from s_{i-1} by picking the left-most non-terminal X in s_{i-1} and replacing it by some β where $X \rightarrow \beta$ is a rule in R.

For example: [S],[NP VP],[Adj Noun VP], [fruit Noun VP], [fruit flies VP],[fruit flies Vb NP],[fruit flies like NP], [fruit flies like Det Noun], [fruit flies like a], [fruit flies like a banana]

Left-most derivation example

S

Left-most derivation example

S
NP VP

$S \rightarrow NP\ VP$

Left-most derivation example

S
NP VP
Adj Noun VP

$\text{NP} \rightarrow \text{Adj Noun}$

Left-most derivation example

S
NP VP
Adj Noun VP
fruit Noun VP

Adj $\rightarrow$ fruit

Left-most derivation example

S
NP VP
Adj Noun VP
fruit Noun VP
fruit flies VP

Noun $\rightarrow$ flies

Left-most derivation example

S
NP VP
Adj Noun VP
fruit Noun VP
fruit flies VP
fruit flies Vb NP

$$\text{VP} \rightarrow \text{Vb NP}$$

Left-most derivation example

S
NP VP
Adj Noun VP
fruit Noun VP
fruit flies VP
fruit flies Vb NP
fruit flies like NP

$$\text{Vb} \rightarrow \text{like}$$

Left-most derivation example

S
NP VP
Adj Noun VP
fruit Noun VP
fruit flies VP
fruit flies Vb NP
fruit flies like NP
fruit flies like Det Noun

$NP \rightarrow$ Det Noun

Left-most derivation example

S
NP VP
Adj Noun VP
fruit Noun VP
fruit flies VP
fruit flies Vb NP
fruit flies like NP
fruit flies like Det Noun
fruit flies like a Noun

$\text{Det} \rightarrow \text{a}$

Left-most derivation example

S
NP VP
Adj Noun VP
fruit Noun VP
fruit flies VP
fruit flies Vb NP
fruit flies like NP
fruit flies like Det Noun
fruit flies like a Noun
fruit flies like a banana

Noun $\rightarrow$ banana

Left-most derivation example

S
NP VP
Adj Noun VP
fruit Noun VP
fruit flies VP
fruit flies Vb NP
fruit flies like NP
fruit flies like Det Noun
fruit flies like a Noun
fruit flies like a banana

- The resulting derivation can be written as a tree.

Left-most derivation example

S
NP VP
Adj Noun VP
fruit Noun VP
fruit flies VP
fruit flies Vb NP
fruit flies like NP
fruit flies like Det Noun
fruit flies like a Noun
fruit flies like a banana

- The resulting derivation can be written as a tree.
- Many trees can be generated.

Context Free Grammars

a simple grammar

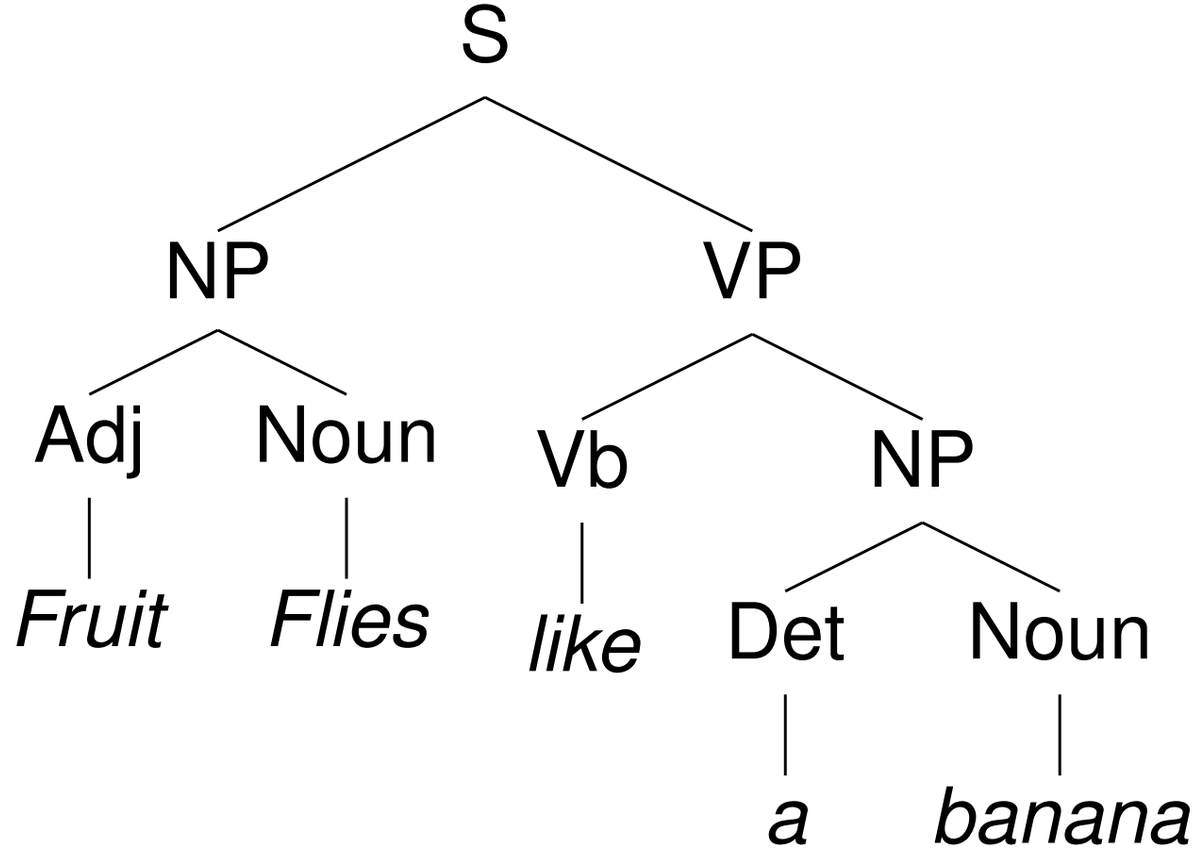
S $\rightarrow$ NP VP
NP $\rightarrow$ Adj Noun
NP $\rightarrow$ Det Noun
VP $\rightarrow$ Vb NP
-
Adj $\rightarrow$ fruit
Noun $\rightarrow$ flies
Vb $\rightarrow$ like
Det $\rightarrow$ a
Noun $\rightarrow$ banana
Noun $\rightarrow$ tomato
Adj $\rightarrow$ angry
. . .

Context Free Grammars

a simple grammar

S $\rightarrow$ NP VP
NP $\rightarrow$ Adj Noun
NP $\rightarrow$ Det Noun
VP $\rightarrow$ Vb NP
-
Adj $\rightarrow$ fruit
Noun $\rightarrow$ flies
Vb $\rightarrow$ like
Det $\rightarrow$ a
Noun $\rightarrow$ banana
Noun $\rightarrow$ tomato
Adj $\rightarrow$ angry
...

Example

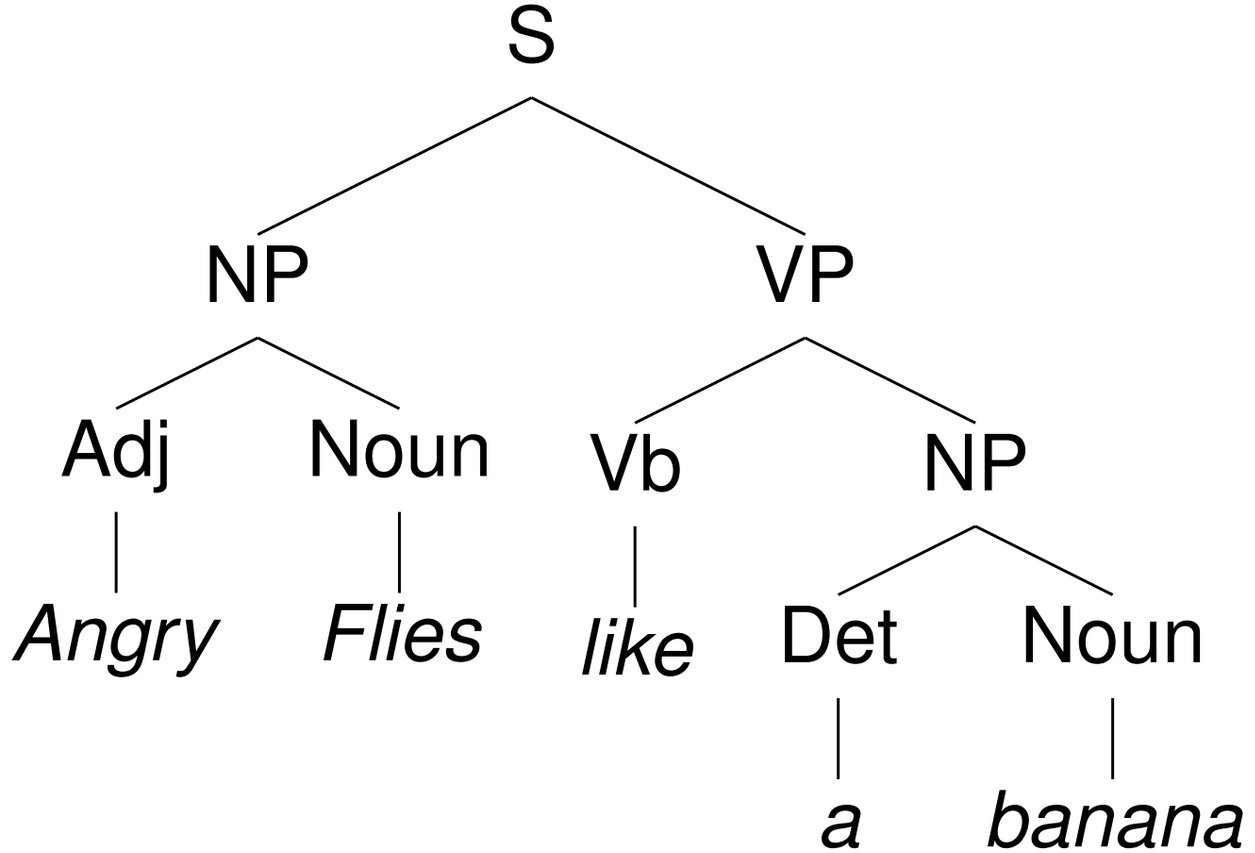


Context Free Grammars

a simple grammar

S $\rightarrow$ NP VP
NP $\rightarrow$ Adj Noun
NP $\rightarrow$ Det Noun
VP $\rightarrow$ Vb NP
-
Adj $\rightarrow$ fruit
Noun $\rightarrow$ flies
Vb $\rightarrow$ like
Det $\rightarrow$ a
Noun $\rightarrow$ banana
Noun $\rightarrow$ tomato
Adj $\rightarrow$ angry
...

Example

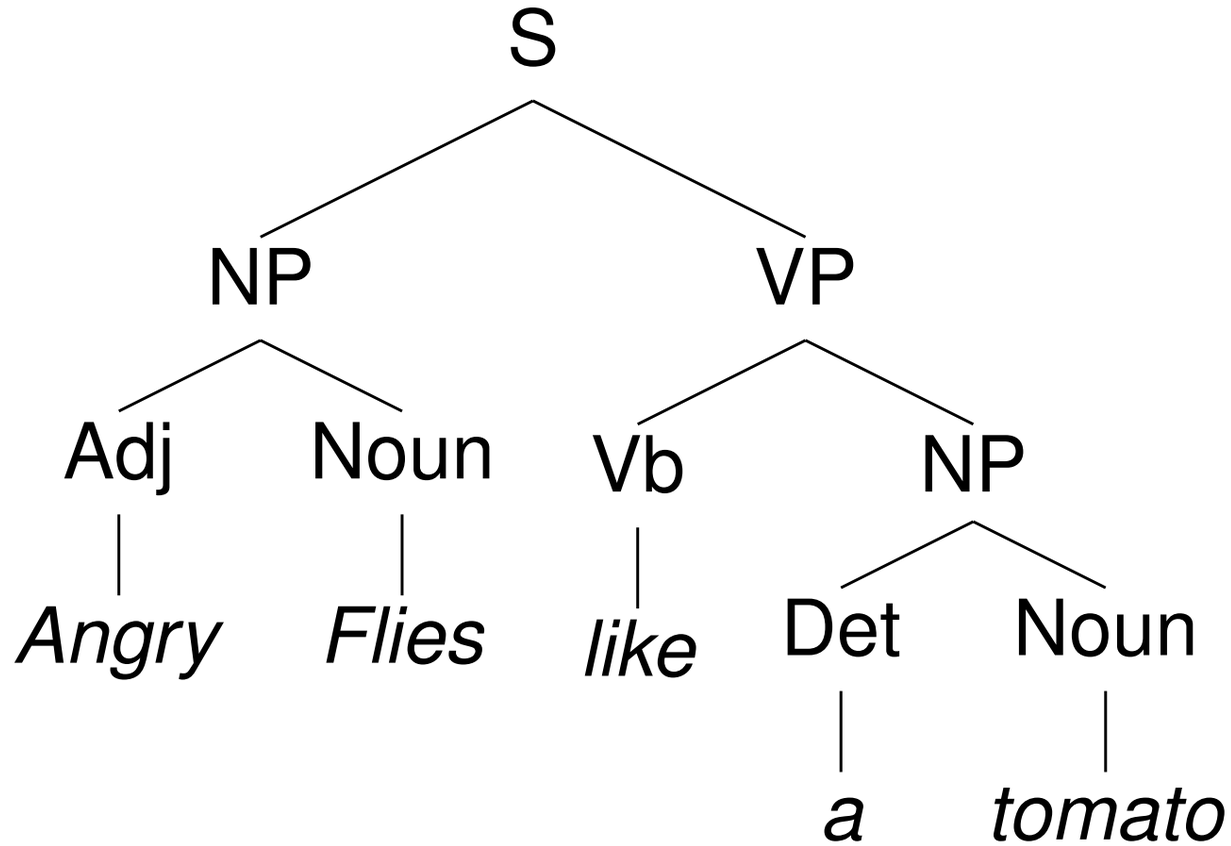


Context Free Grammars

a simple grammar

S $\rightarrow$ NP VP
NP $\rightarrow$ Adj Noun
NP $\rightarrow$ Det Noun
VP $\rightarrow$ Vb NP
-
Adj $\rightarrow$ fruit
Noun $\rightarrow$ flies
Vb $\rightarrow$ like
Det $\rightarrow$ a
Noun $\rightarrow$ banana
Noun $\rightarrow$ tomato
Adj $\rightarrow$ angry
...

Example

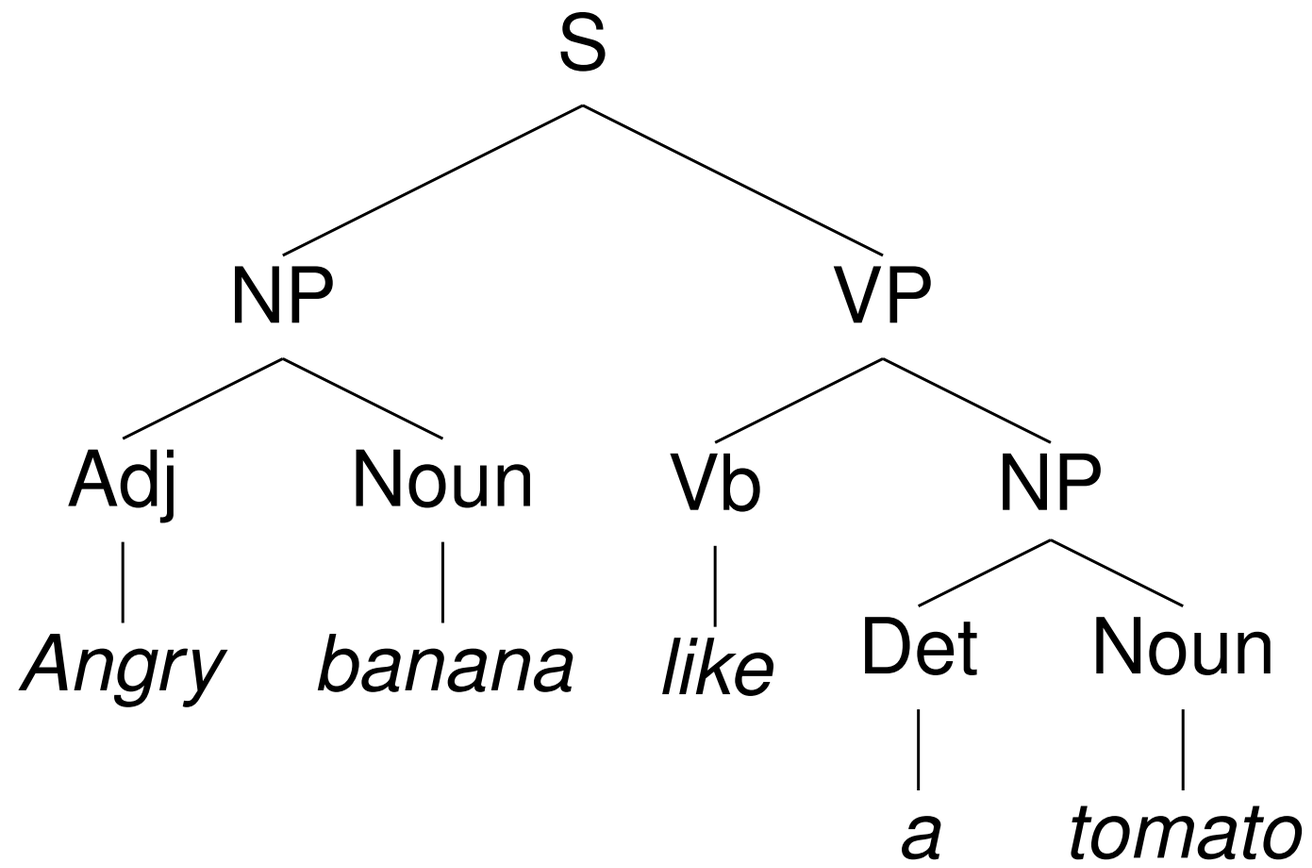


Context Free Grammars

a simple grammar

S $\rightarrow$ NP VP
NP $\rightarrow$ Adj Noun
NP $\rightarrow$ Det Noun
VP $\rightarrow$ Vb NP
-
Adj $\rightarrow$ fruit
Noun $\rightarrow$ flies
Vb $\rightarrow$ like
Det $\rightarrow$ a
Noun $\rightarrow$ banana
Noun $\rightarrow$ tomato
Adj $\rightarrow$ angry
...

Example

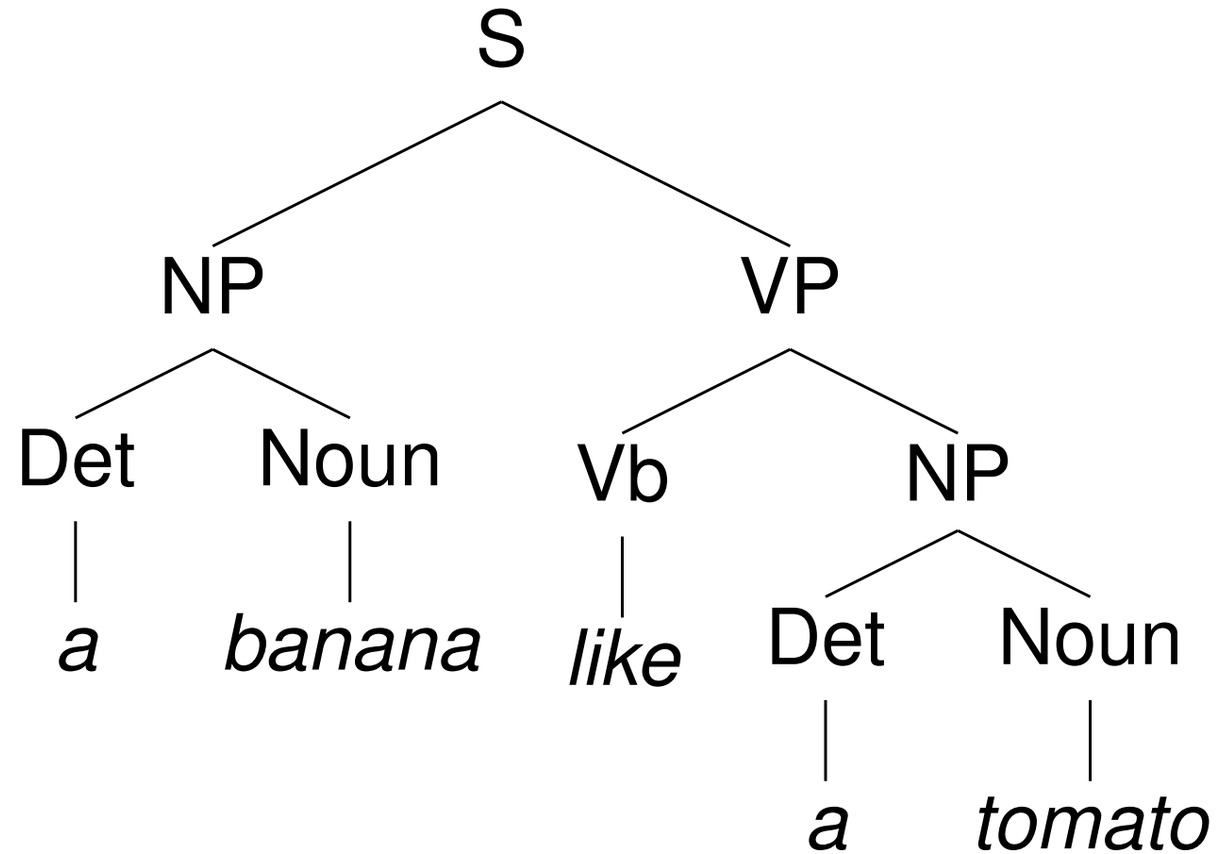


Context Free Grammars

a simple grammar

S $\rightarrow$ NP VP
NP $\rightarrow$ Adj Noun
NP $\rightarrow$ Det Noun
VP $\rightarrow$ Vb NP
-
Adj $\rightarrow$ fruit
Noun $\rightarrow$ flies
Vb $\rightarrow$ like
Det $\rightarrow$ a
Noun $\rightarrow$ banana
Noun $\rightarrow$ tomato
Adj $\rightarrow$ angry
...

Example

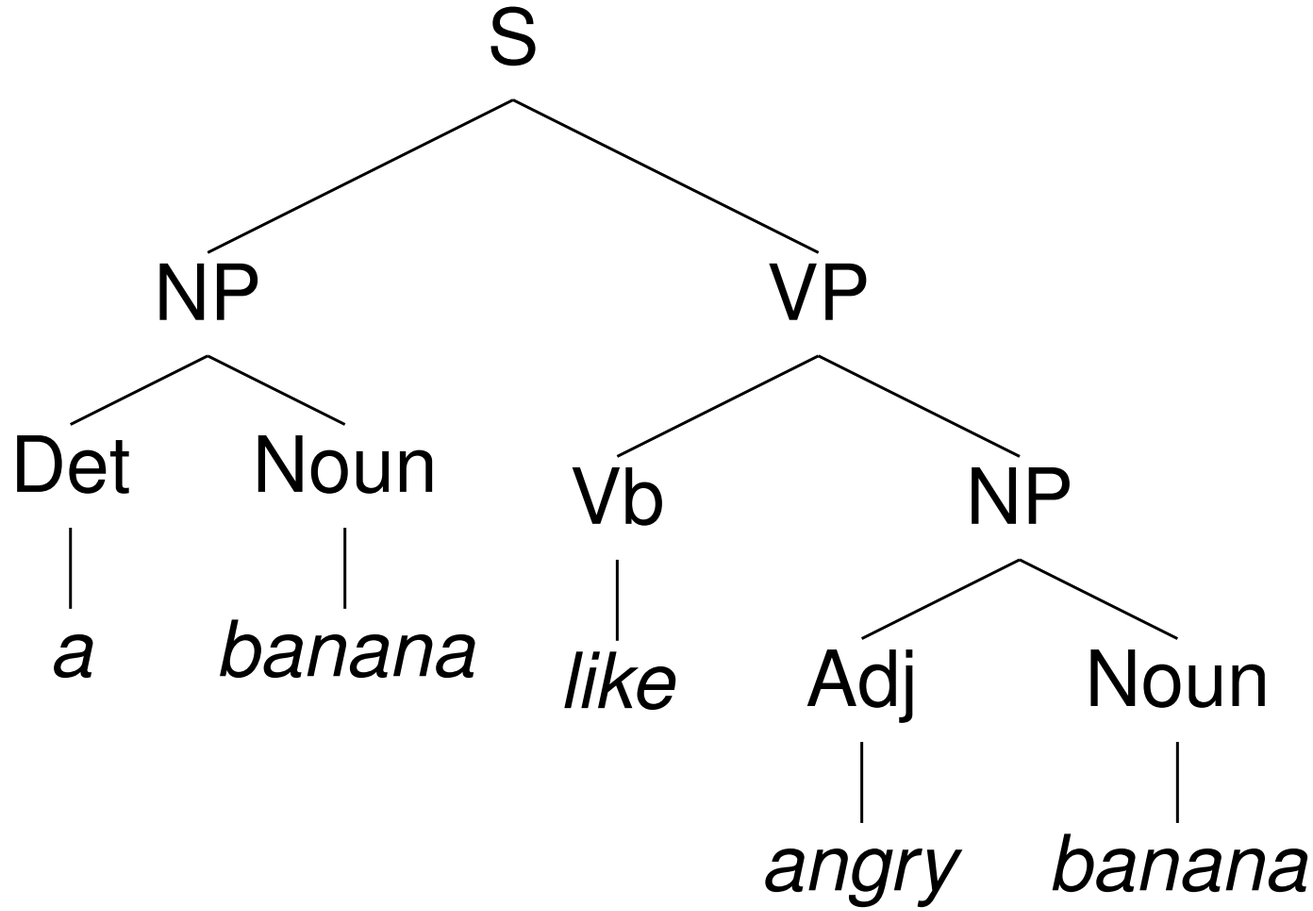


Context Free Grammars

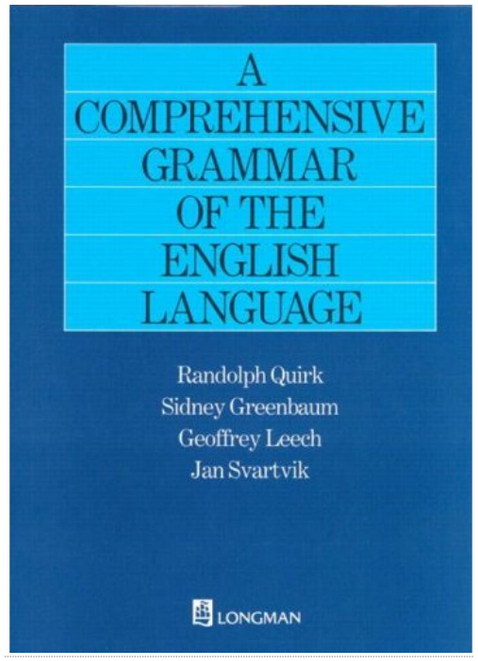
a simple grammar

S $\rightarrow$ NP VP
NP $\rightarrow$ Adj Noun
NP $\rightarrow$ Det Noun
VP $\rightarrow$ Vb NP
-
Adj $\rightarrow$ fruit
Noun $\rightarrow$ flies
Vb $\rightarrow$ like
Det $\rightarrow$ a
Noun $\rightarrow$ banana
Noun $\rightarrow$ tomato
Adj $\rightarrow$ angry
...

Example



A Brief Introduction to English Syntax



Product Details (from Amazon)
Hardcover: 1779 pages
Publisher: Longman; 2nd Revised edition
Language: English
ISBN-10: 0582517346
ISBN-13: 978-0582517349
Product Dimensions: 8.4 x 2.4 x 10 inches
Shipping Weight: 4.6 pounds

(by Mike Collins)

A Brief Overview of English Syntax

Parts of Speech (tags from the Brown corpus):

- Nouns
  NN = singular noun e.g., man, dog, park
  NNS = plural noun e.g., telescopes, houses, buildings
  NNP = proper noun e.g., Smith, Gates, IBM
- Determiners
  DT = determiner e.g., the, a, some, every
- Adjectives
  JJ = adjective e.g., red, green, large, idealistic

A Fragment of a Noun Phrase Grammar

$\bar{N}$	$\Rightarrow$	NN	
$\bar{N}$	$\Rightarrow$	NN	$\bar{N}$
$\bar{N}$	$\Rightarrow$	JJ	$\bar{N}$
$\bar{N}$	$\Rightarrow$	$\bar{N}$	$\bar{N}$
NP	$\Rightarrow$	DT	$\bar{N}$

NN	$\Rightarrow$	box
NN	$\Rightarrow$	car
NN	$\Rightarrow$	mechanic
NN	$\Rightarrow$	pigeon
DT	$\Rightarrow$	the
DT	$\Rightarrow$	a

JJ	$\Rightarrow$	fast
JJ	$\Rightarrow$	metal
JJ	$\Rightarrow$	idealistic
JJ	$\Rightarrow$	clay

the box
a car
the pigeon car
the fast metal pigeon

(by Mike Collins)

Prepositions, and Prepositional Phrases

- Prepositions

IN = preposition e.g., of, in, out, beside, as

(by Mike Collins)

An Extended Grammar

$\bar{N}$	$\Rightarrow$	NN	
$\bar{N}$	$\Rightarrow$	NN	$\bar{N}$
$\bar{N}$	$\Rightarrow$	JJ	$\bar{N}$
$\bar{N}$	$\Rightarrow$	$\bar{N}$	$\bar{N}$
NP	$\Rightarrow$	DT	$\bar{N}$
PP	$\Rightarrow$	IN	NP
$\bar{N}$	$\Rightarrow$	$\bar{N}$	PP

(by Mike Collins)

An Extended Grammar

$\bar{N}$	$\Rightarrow$	NN	
$\bar{N}$	$\Rightarrow$	NN	$\bar{N}$
$\bar{N}$	$\Rightarrow$	JJ	$\bar{N}$
$\bar{N}$	$\Rightarrow$	$\bar{N}$	$\bar{N}$
NP	$\Rightarrow$	DT	$\bar{N}$
PP	$\Rightarrow$	IN	NP
$\bar{N}$	$\Rightarrow$	$\bar{N}$	PP

NN	$\Rightarrow$	box
NN	$\Rightarrow$	car
NN	$\Rightarrow$	mechanic
NN	$\Rightarrow$	pigeon
DT	$\Rightarrow$	the
DT	$\Rightarrow$	a

JJ	$\Rightarrow$	fast
JJ	$\Rightarrow$	metal
JJ	$\Rightarrow$	idealistic
JJ	$\Rightarrow$	clay
IN	$\Rightarrow$	in
IN	$\Rightarrow$	under
IN	$\Rightarrow$	of
IN	$\Rightarrow$	on
IN	$\Rightarrow$	with
IN	$\Rightarrow$	as

Generates:

in a box, under the box, the fast car mechanic under the pigeon in the box, ...

(by Mike Collins)

Verbs, Verb Phrases, and Sentences

- Basic Verb Types
  Vi = Intransitive verb e.g., sleeps, walks, laughs
  Vt = Transitive verb e.g., sees, saw, likes
  Vd = Ditransitive verb e.g., gave

- Basic VP Rules
  VP → Vi
  VP → Vt NP
  VP → Vd NP NP

- Basic S Rule
  S → NP VP

Examples of VP:
sleeps, walks, likes the mechanic, gave the mechanic the fast car
Examples of S:
the man sleeps, the dog walks, the dog gave the mechanic the fast car

(by Mike Collins)

PPs Modifying Verb Phrases

A new rule: VP $\rightarrow$ VP PP

New examples of VP:
sleeps in the car, walks like the mechanic, gave the mechanic the fast car on Tuesday, ...

(by Mike Collins)

Complementizers, and SBARs

- Complementizers
  COMP = complementizer e.g., that
- SBAR
  SBAR → COMP S

Examples:

that the man sleeps, that the mechanic saw the dog . . .

(by Mike Collins)

More Verbs

- New Verb Types
  V[5] e.g., said, reported
  V[6] e.g., told, informed
  V[7] e.g., bet

- New VP Rules

VP	$\rightarrow$	V[5]	SBAR		
VP	$\rightarrow$	V[6]	NP	SBAR	
VP	$\rightarrow$	V[7]	NP	NP	SBAR

Examples of New VPs:

said that the man sleeps

told the dog that the mechanic likes the pigeon

bet the pigeon $50 that the mechanic owns a fast car

(by Mike Collins)

Coordination

- A New Part-of-Speech:
  CC = Coordinator e.g., and, or, but
- New Rules

NP	$\rightarrow$	NP	CC	NP
$\bar{\text{N}}$	$\rightarrow$	$\bar{\text{N}}$	CC	$\bar{\text{N}}$
VP	$\rightarrow$	VP	CC	VP
S	$\rightarrow$	S	CC	S
SBAR	$\rightarrow$	SBAR	CC	SBAR

(by Mike Collins)

We've Only Scratched the Surface...

- Agreement

  The dogs laugh *vs.* The dog laughs

- Wh-movement

  The dog that the cat liked ___

- Active vs. passive

  The dog saw the cat *vs.*
  The cat was seen by the dog

- If you're interested in reading more:

  Syntactic Theory: A Formal Introduction, 2nd Edition. Ivan A. Sag, Thomas Wasow, and Emily M. Bender.

(by Mike Collins)

CFGs and center embedding

The mouse ate the corn.
The mouse that the snake ate ate the corn.
The mouse that the snake that the hawk ate ate ate the corn.

....

These sentences are all grammatical.
They can be generated by a CFG:

S → NP VP
NP → NP RelClause
RelClause → that NP ate

Linguists distinguish between a speaker's
- competence (grammatical knowledge) **and**
- performance (processing and memory limitations)

Parsing with (P)CFGs

Parsing with CFGs

Let's assume...

- Let's assume natural language is generated by a CFG.
- ...and let's assume we have the grammar.
- Then parsing is easy: given a sentence, find the chain of derivations starting from S that generates it.

Parsing with CFGs

Let's assume...

- Let's assume natural language is generated by a CFG.
- ...and let's assume we have the grammar.
- Then parsing is easy: given a sentence, find the chain of derivations starting from S that generates it.

Problem

- Natural Language is NOT generated by a CFG.

Parsing with CFGs

Let's assume...

- Let's assume natural language is generated by a CFG.
- ...and let's assume we have the grammar.
- Then parsing is easy: given a sentence, find the chain of derivations starting from S that generates it.

Problem

- Natural Language is NOT generated by a CFG.

Solution

- We assume really hard that it is.

Parsing with CFGs

Let's assume...

- Let's assume natural language is generated by a CFG.
- ...and let's assume we have the grammar.
- Then parsing is easy: given a sentence, find the chain of derivations starting from S that generates it.

Problem

- We don't have the grammar.

Parsing with CFGs

Let's assume...

- Let's assume natural language is generated by a CFG.
- ...and let's assume we have the grammar.
- Then parsing is easy: given a sentence, find the chain of derivations starting from S that generates it.

Problem

- We don't have the grammar.

Solution

- We'll ask a genius linguist to write it!

Parsing with CFGs

Let's assume...

- Let's assume natural language is generated by a CFG.
- ...and let's assume we have the grammar.
- Then parsing is easy: given a sentence, find the chain of derivations starting from S that generates it.

Problem

- How do we find the chain of derivations?

Parsing with CFGs

Let's assume...

- Let's assume natural language is generated by a CFG.
- ...and let's assume we have the grammar.
- Then parsing is easy: given a sentence, find the chain of derivations starting from S that generates it.

Problem

- How do we find the chain of derivations?

Solution

- With dynamic programming! (soon)

Parsing with CFGs

Let's assume...

- Let's assume natural language is generated by a CFG.
- ...and let's assume we have the grammar.
- Then parsing is easy: given a sentence, find the chain of derivations starting from S that generates it.

Problem

- Real grammar: hundreds of possible derivations per sentence.

Parsing with CFGs

Let's assume...

- Let's assume natural language is generated by a CFG.
- ...and let's assume we have the grammar.
- Then parsing is easy: given a sentence, find the chain of derivations starting from S that generates it.

Problem

- Real grammar: hundreds of possible derivations per sentence.

Solution

- No problem! We'll choose the best one. (sooner)

Obtaining a Grammar

Let a genius linguist write it

- Hard. Many rules, many complex interactions.
- Genius linguists don’t grow on trees !

Obtaining a Grammar

Let a genius linguist write it

- Hard. Many rules, many complex interactions.
- Genius linguists don't grow on trees !

An easier way - ask a linguist to grow trees

- Ask a linguist to annotate sentences with tree structure.
- (This need not be a genius – Smart is enough.)
- Then extract the rules from the annotated trees.

Obtaining a Grammar

Let a genius linguist write it

- Hard. Many rules, many complex interactions.
- Genius linguists don't grow on trees !

An easier way - ask a linguist to grow trees

- Ask a linguist to annotate sentences with tree structure.
- (This need not be a genius – Smart is enough.)
- Then extract the rules from the annotated trees.

Treebanks

- **English Treebank**: 40k sentences, manually annotated with tree structure.
- **Hebrew Treebank**: about 5k sentences

Treebank Sentence Example

```
( (S
    (NP-SBJ
      (NP (NNP Pierre) (NNP Vinken) )
      (, ,)
      (ADJP
        (NP (CD 61) (NNS years) )
        (JJ old) )
      (, ,) )
    (VP (MD will)
      (VP (VB join)
        (NP (DT the) (NN board) )
        (PP-CLR (IN as)
          (NP (DT a) (JJ nonexecutive) (NN director
        (NP-TMP (NNP Nov.) (CD 29) )))
    (. .) ))
```

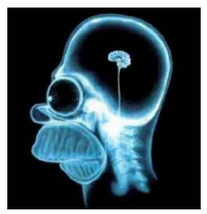
Supervised Learning from a Treebank

((fruit/ADJ flies/NN) (like/VB

(time/NN (flies/VB (like/IN

. . . . . .

. . . . . .



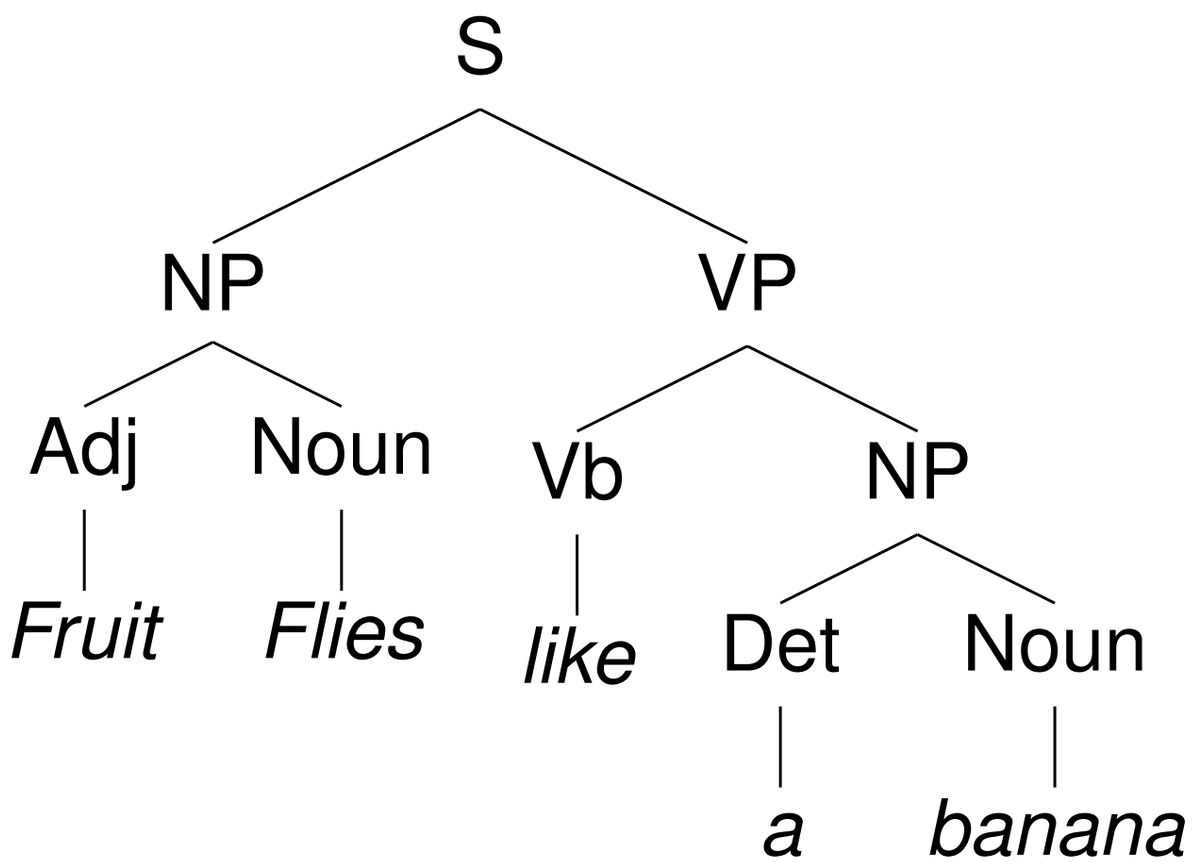
Extracting CFG from Trees

- The leafs of the trees define Σ
- The internal nodes of the trees define N
- Add a special S symbol on top of all trees
- Each node an its children is a rule in R

Extracting CFG from Trees

- The leafs of the trees define Σ
- The internal nodes of the trees define N
- Add a special S symbol on top of all trees
- Each node an its children is a rule in R

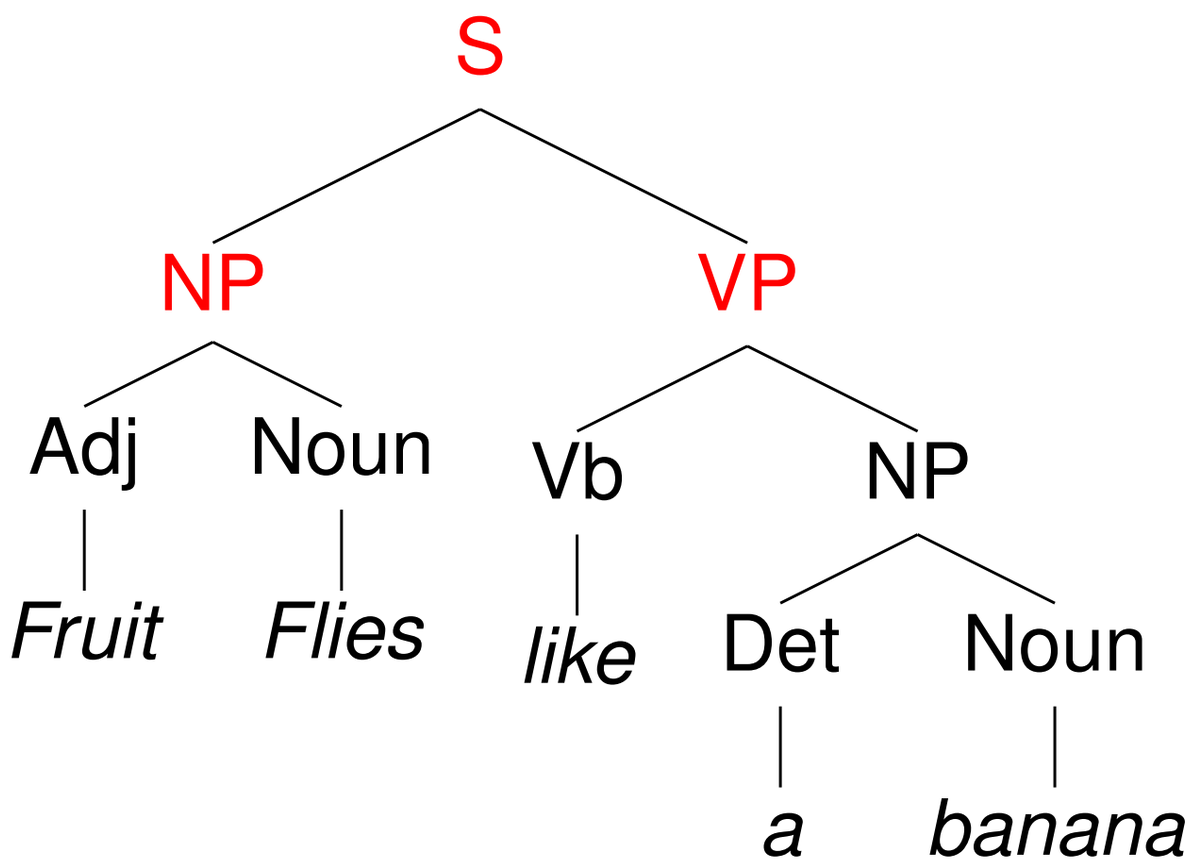
Extracting Rules



Extracting CFG from Trees

- The leafs of the trees define Σ
- The internal nodes of the trees define N
- Add a special S symbol on top of all trees
- Each node an its children is a rule in R

Extracting Rules

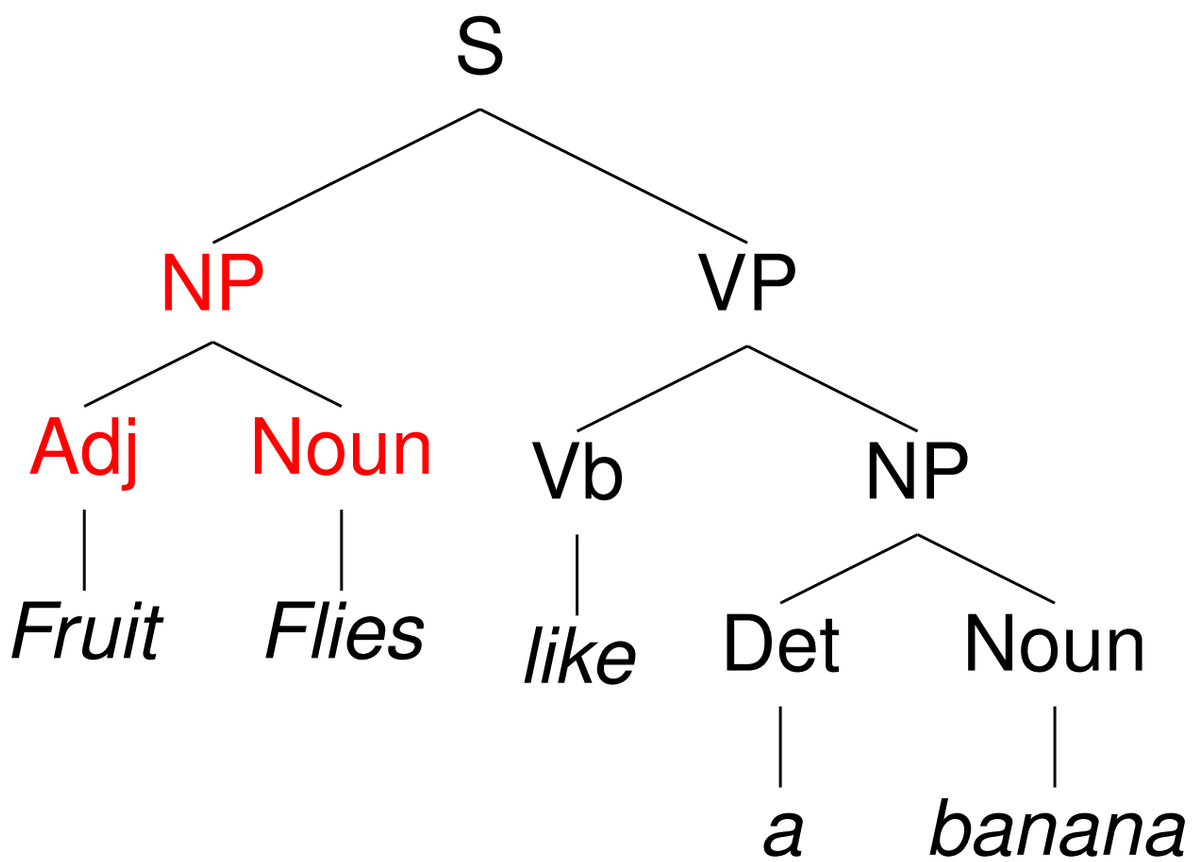


S $\rightarrow$ **NP VP**

Extracting CFG from Trees

- The leafs of the trees define Σ
- The internal nodes of the trees define N
- Add a special S symbol on top of all trees
- Each node an its children is a rule in R

Extracting Rules



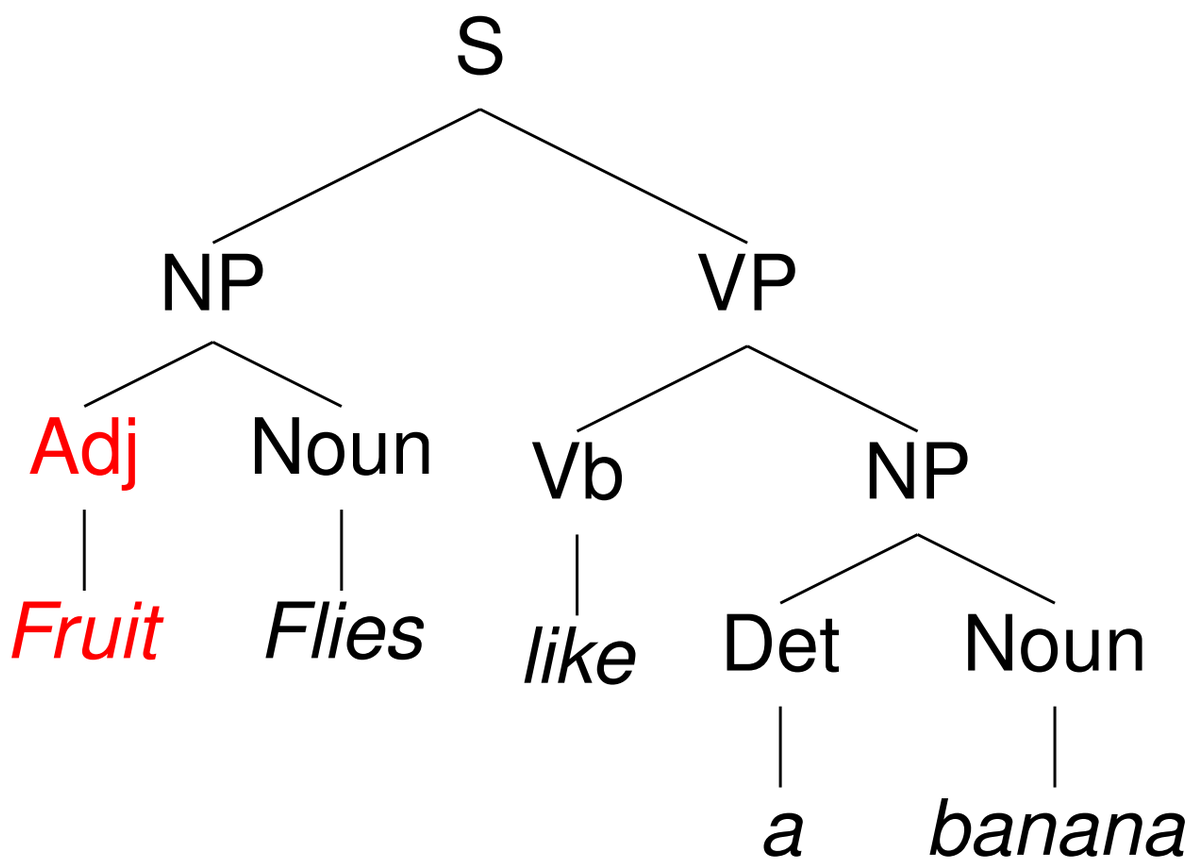
$S \rightarrow NP\ VP$

NP $\rightarrow$ Adj Noun

Extracting CFG from Trees

- The leafs of the trees define Σ
- The internal nodes of the trees define N
- Add a special S symbol on top of all trees
- Each node an its children is a rule in R

Extracting Rules



$S \rightarrow NP\ VP$

$NP \rightarrow Adj\ Noun$

Adj $\rightarrow$ **fruit**

From CFG to PCFG

- English is NOT generated from CFG $\Rightarrow$ It's generated by a PCFG!

From CFG to PCFG

- English is NOT generated from CFG $\Rightarrow$ It's generated by a PCFG!
- PCFG: probabilistic context free grammar. Just like a CFG, but each rule has an associated probability.
- All probabilities for the same LHS sum to 1.

From CFG to PCFG

- English is NOT generated from CFG $\Rightarrow$ It's generated by a PCFG!
- PCFG: probabilistic context free grammar. Just like a CFG, but each rule has an associated probability.
- All probabilities for the same LHS sum to 1.
- Multiplying all the rule probs in a derivation gives the probability of the derivation.
- We want the tree with maximum probability.

From CFG to PCFG

- English is NOT generated from CFG $\Rightarrow$ It's generated by a PCFG!
- PCFG: probabilistic context free grammar. Just like a CFG, but each rule has an associated probability.
- All probabilities for the same LHS sum to 1.
- Multiplying all the rule probs in a derivation gives the probability of the derivation.
- We want the tree with maximum probability.

More Formally

$$P(tree, sent) = \prod_{l \to r \in deriv(tree)} p(l \to r)$$

From CFG to PCFG

- English is NOT generated from CFG $\Rightarrow$ It's generated by a PCFG!
- PCFG: probabilistic context free grammar. Just like a CFG, but each rule has an associated probability.
- All probabilities for the same LHS sum to 1.
- Multiplying all the rule probs in a derivation gives the probability of the derivation.
- We want the tree with maximum probability.

More Formally

$$P(tree, sent) = \prod_{l \to r \in deriv(tree)} p(l \to r)$$

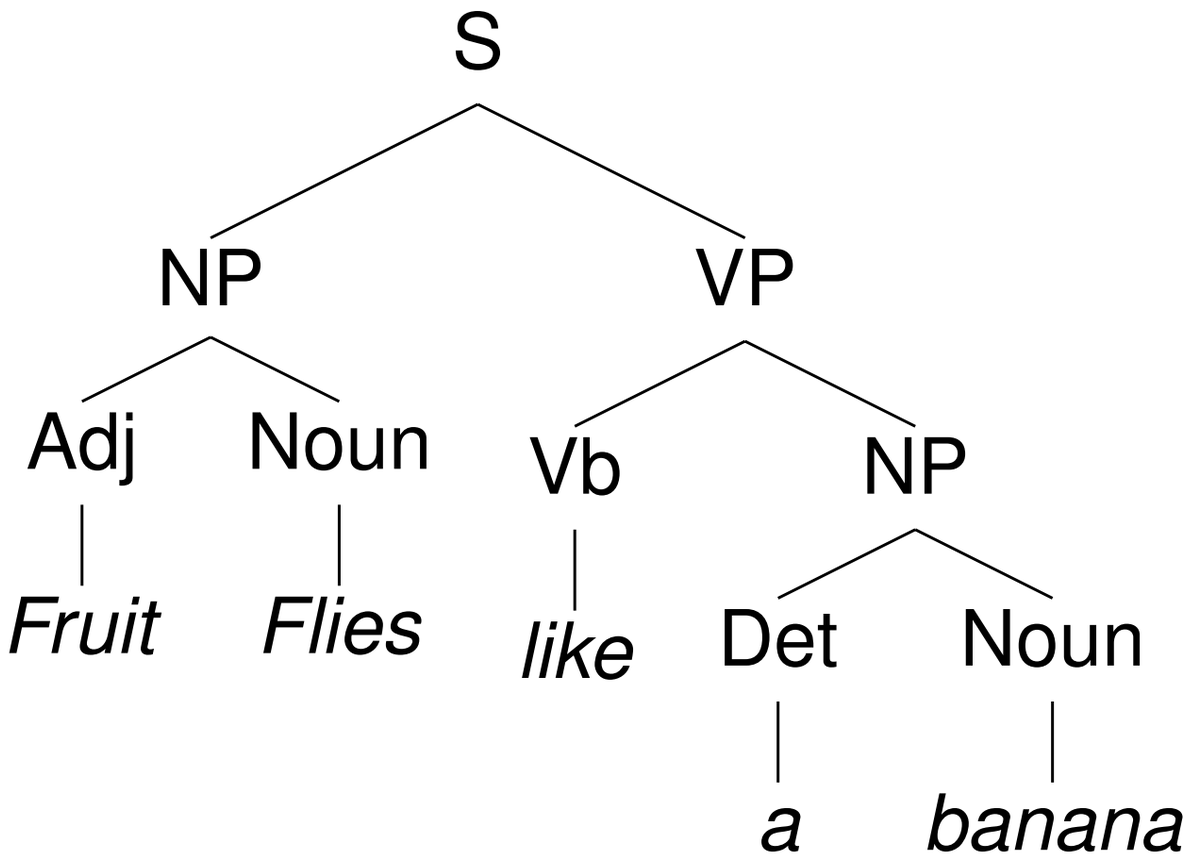
$$tree = \arg\max_{tree \in trees(sent)} P(tree|sent) = \arg\max_{tree \in trees(sent)} P(tree, sent)$$

PCFG Example

a simple PCFG

1.0 S $\rightarrow$ NP VP
0.3 NP $\rightarrow$ Adj Noun
0.7 NP $\rightarrow$ Det Noun
1.0 VP $\rightarrow$ Vb NP
-
0.2 Adj $\rightarrow$ fruit
0.2 Noun $\rightarrow$ flies
1.0 Vb $\rightarrow$ like
1.0 Det $\rightarrow$ a
0.4 Noun $\rightarrow$ banana
0.4 Noun $\rightarrow$ tomato
0.8 Adj $\rightarrow$ angry

Example



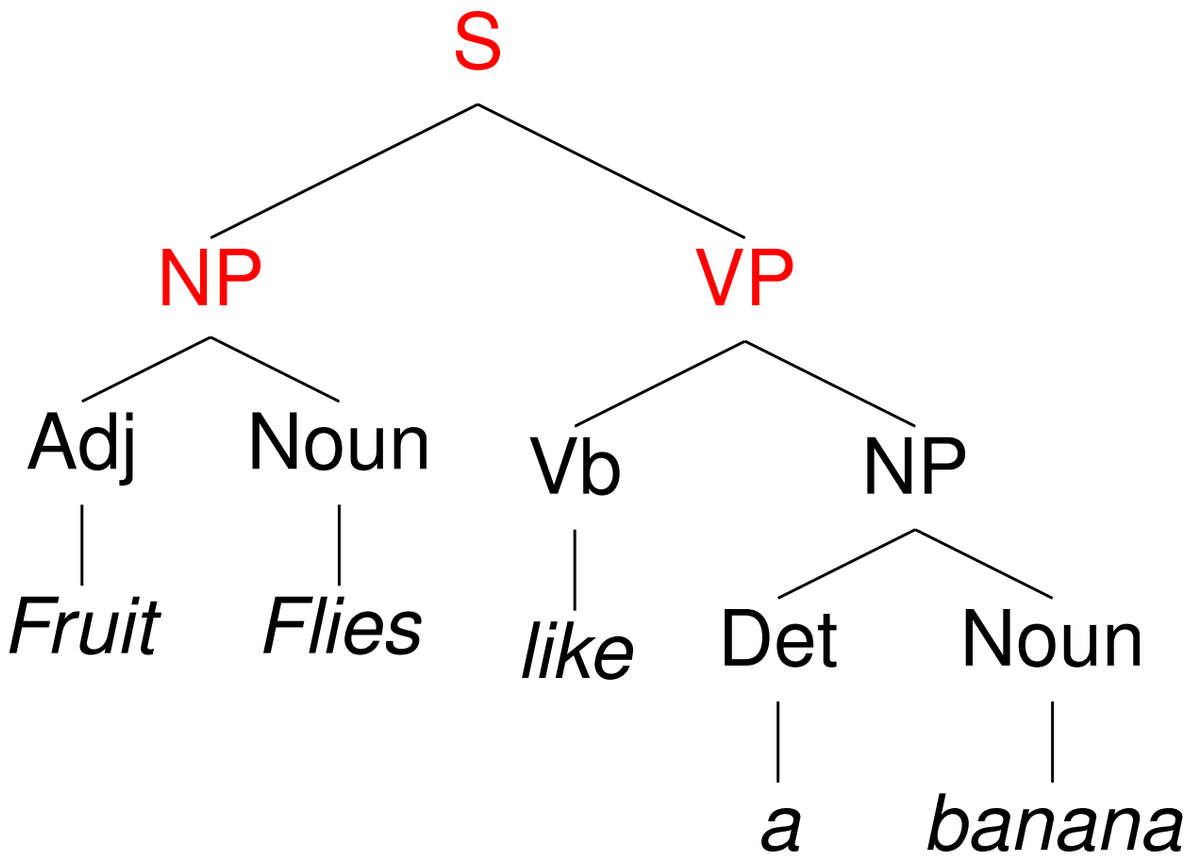
$1 * 0.3 * 0.2 * 0.7 * 1.0 * 0.2 * 1 * 1 * 0.4 = 0.0033$

PCFG Example

a simple PCFG

1.0 S $\rightarrow$ NP VP
0.3 NP $\rightarrow$ Adj Noun
0.7 NP $\rightarrow$ Det Noun
1.0 VP $\rightarrow$ Vb NP
-
0.2 Adj $\rightarrow$ fruit
0.2 Noun $\rightarrow$ flies
1.0 Vb $\rightarrow$ like
1.0 Det $\rightarrow$ a
0.4 Noun $\rightarrow$ banana
0.4 Noun $\rightarrow$ tomato
0.8 Adj $\rightarrow$ angry

Example



$1 * 0.3 * 0.2 * 0.7 * 1.0 * 0.2 * 1 * 1 * 0.4 = 0.0033$

PCFG Example

a simple PCFG

1.0 S $\rightarrow$ NP VP

0.3 NP $\rightarrow$ Adj Noun

0.7 NP $\rightarrow$ Det Noun

1.0 VP $\rightarrow$ Vb NP

-

0.2 Adj $\rightarrow$ fruit

0.2 Noun $\rightarrow$ flies

1.0 Vb $\rightarrow$ like

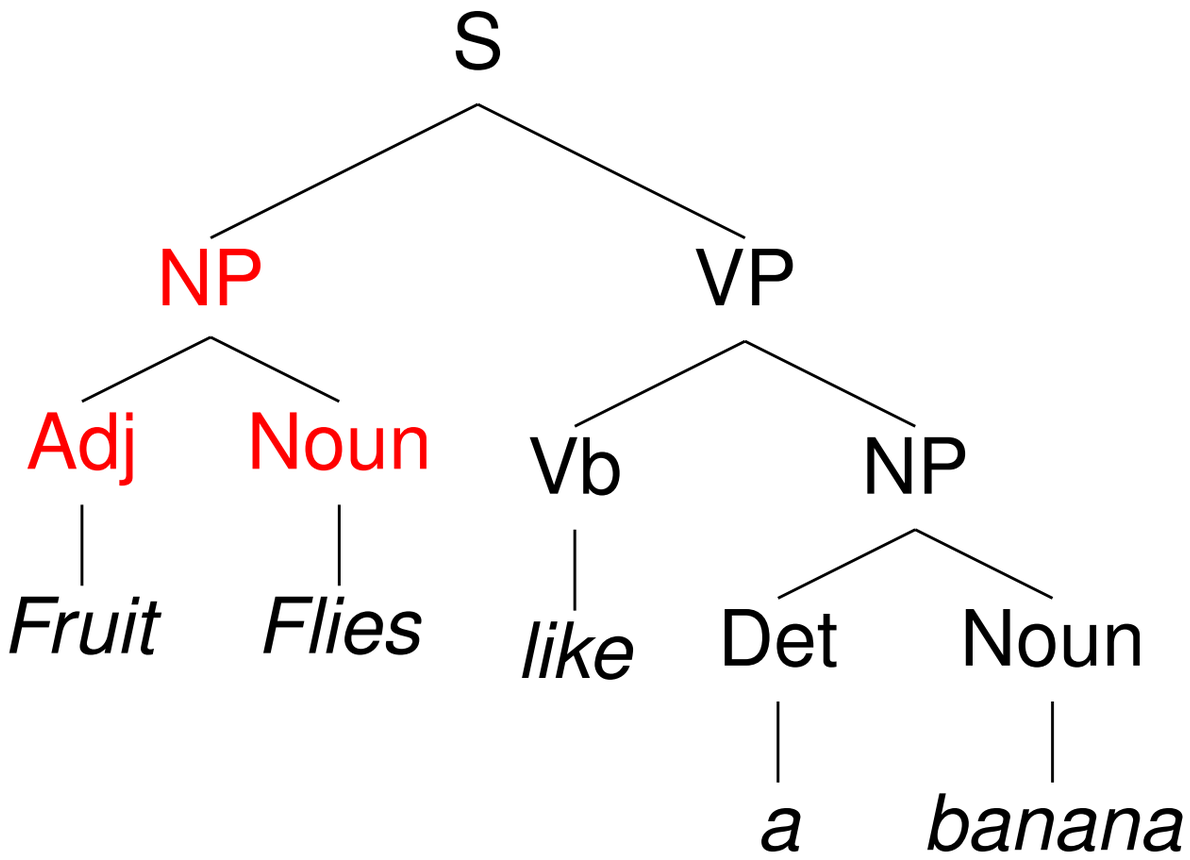
1.0 Det $\rightarrow$ a

0.4 Noun $\rightarrow$ banana

0.4 Noun $\rightarrow$ tomato

0.8 Adj $\rightarrow$ angry

Example



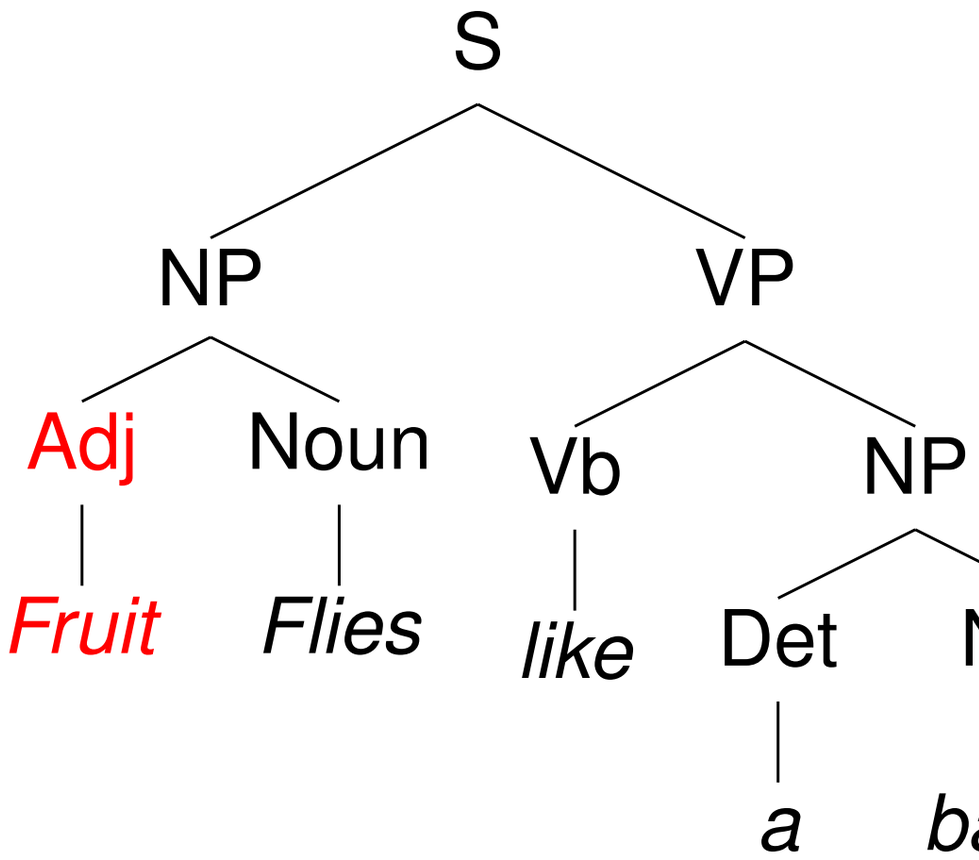
$1 * 0.3 * 0.2 * 0.7 * 1.0 * 0.2 * 1 * 1 * 0.4 =$
0.0033

PCFG Example

a simple PCFG

1.0 S $\rightarrow$ NP VP
0.3 NP $\rightarrow$ Adj Noun
0.7 NP $\rightarrow$ Det Noun
1.0 VP $\rightarrow$ Vb NP
-
0.2 Adj $\rightarrow$ fruit
0.2 Noun $\rightarrow$ flies
1.0 Vb $\rightarrow$ like
1.0 Det $\rightarrow$ a
0.4 Noun $\rightarrow$ banana
0.4 Noun $\rightarrow$ tomato
0.8 Adj $\rightarrow$ angry

Example



$1 * 0.3 * 0.2 * 0.7 * 1.0 * 0.2 * 1 * 1 * 0.4 = 0.0033$

Parsing with PCFG

- Parsing with a PCFG is finding the most probable derivation for a given sentence.
- This can be done quite efficiently with dynamic programming (the CKY algorithm)

Parsing with PCFG

- Parsing with a PCFG is finding the most probable derivation for a given sentence.
- This can be done quite efficiently with dynamic programming (the CKY algorithm)

Obtaining the probabilities

- We estimate them from the Treebank.
- $P(LHS \rightarrow RHS) = \frac{count(LHS \rightarrow RHS)}{count(LHS \rightarrow \Diamond)}$
- We can also add smoothing and backoff, as before.
- Dealing with unknown words - like in the HMM

The CKY algorithm

The Problem

Input

- Sentence (a list of words)
  - n – sentence length
- CFG Grammar (with weights on rules)
  - g – number of non-terminal symbols

Output

- A parse tree / the best parse tree

But. . .

- Exponentially many possible parse trees!

Solution

- Dynamic Programming!

CKY

Cocke Kasami Younger

CKY

Cocke Kasami Younger
196?

CKY

Cocke 196? Kasami 1965 Younger

CKY

Cocke	Kasami	Younger
196?	1965	1967

3 Interesting Problems

- Recognition
- Parsing
- Disambiguation

3 Interesting Problems

- Recognition
  - Can this string be generated by the grammar?
- Parsing

- Disambiguation

3 Interesting Problems

- Recognition
  - Can this string be generated by the grammar?
- Parsing
  - Show me a possible derivation. . .
- Disambiguation

3 Interesting Problems

- Recognition
  - Can this string be generated by the grammar?
- Parsing
  - Show me a possible derivation. . .
- Disambiguation
  - Show me THE BEST derivation

3 Interesting Problems

- Recognition
  - Can this string be generated by the grammar?
- Parsing
  - Show me a possible derivation. . .
- Disambiguation
  - Show me THE BEST derivation

CKY can do all of these in polynomial time

3 Interesting Problems

- Recognition
  - Can this string be generated by the grammar?
- Parsing
  - Show me a possible derivation. . .
- Disambiguation
  - Show me THE BEST derivation

CKY can do all of these in polynomial time

- For any **CNF** grammar

CNF

Chomsky Normal Form

Definition

A CFG is in CNF form if it only has rules like:

- $A \to B\ C$
- $A \to \alpha$

A,B,C are non terminal symbols

α is a terminal symbol (a word. . .)

- All terminal symbols are RHS of unary rules
- All non terminal symbols are RHS of **binary** rules

CKY can be easily extended to handle also unary rules: $A \to B$

Binarization

Fact

- Any CFG grammar can be converted to CNF form

Binarization

Fact

- Any CFG grammar can be converted to CNF form

Speficifally for Natural Language grammars

- We already have $A \rightarrow \alpha$

Binarization

Fact

- Any CFG grammar can be converted to CNF form

Speficifally for Natural Language grammars

- We already have $A \rightarrow \alpha$
  - ($A \rightarrow \alpha\ \beta$ is also easy to handle)

Binarization

Fact

- Any CFG grammar can be converted to CNF form

Speficifally for Natural Language grammars

- We already have $A \rightarrow \alpha$
  - ($A \rightarrow \alpha\ \beta$ is also easy to handle)
- Unary rules ($A \rightarrow B$) are OK

Binarization

Fact

- Any CFG grammar can be converted to CNF form

Speficifally for Natural Language grammars

- We already have $A \rightarrow \alpha$
  - ($A \rightarrow \alpha\ \beta$ is also easy to handle)
- Unary rules ($A \rightarrow B$) are OK
- Only problem:$S \rightarrow NP\ PP\ VP\ PP$

Binarization

Fact

- Any CFG grammar can be converted to CNF form

Speficifally for Natural Language grammars

- We already have $A \rightarrow \alpha$
  - ($A \rightarrow \alpha\ \beta$ is also easy to handle)
- Unary rules ($A \rightarrow B$) are OK
- Only problem:$S \rightarrow NP\ PP\ VP\ PP$

Binarization

$$
\begin{array}{lcl}
S & \rightarrow & NP\ NP|PP.VP.PP \\
NP|PP.VP.PP & \rightarrow & PP\ NP.PP|VP.PP \\
NP.PP|VP.PP & \rightarrow & VP\ NP.PP.VP|PP
\end{array}
$$

Finally, CKY

Recognition

- Main idea:
  - Build parse tree from bottom up
  - Combine built trees to form bigger trees using grammar rules
  - When left with a single tree, verify root is S
- Exponentially many possible trees. . .
  - Search over all of them in polynomial time using DP
  - Shared structure – smaller trees

Main Idea

If we know:

- $w_i \ldots w_j$ is an NP
- $w_{j+1} \ldots w_k$ is a VP

and grammar has rule:

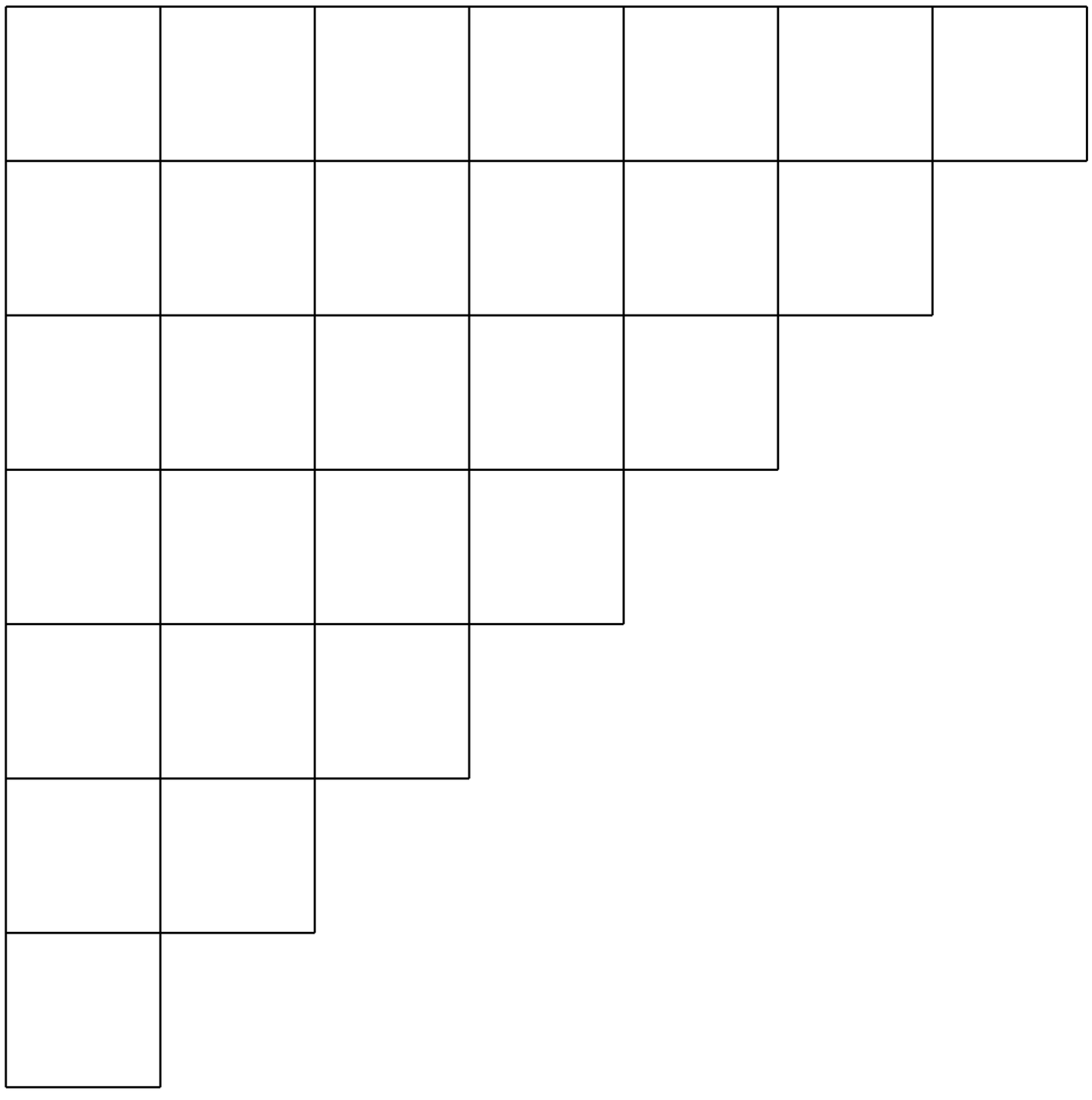
- $S \rightarrow NP\ VP$

Then we know:

- S can derive $w_i \ldots w_k$

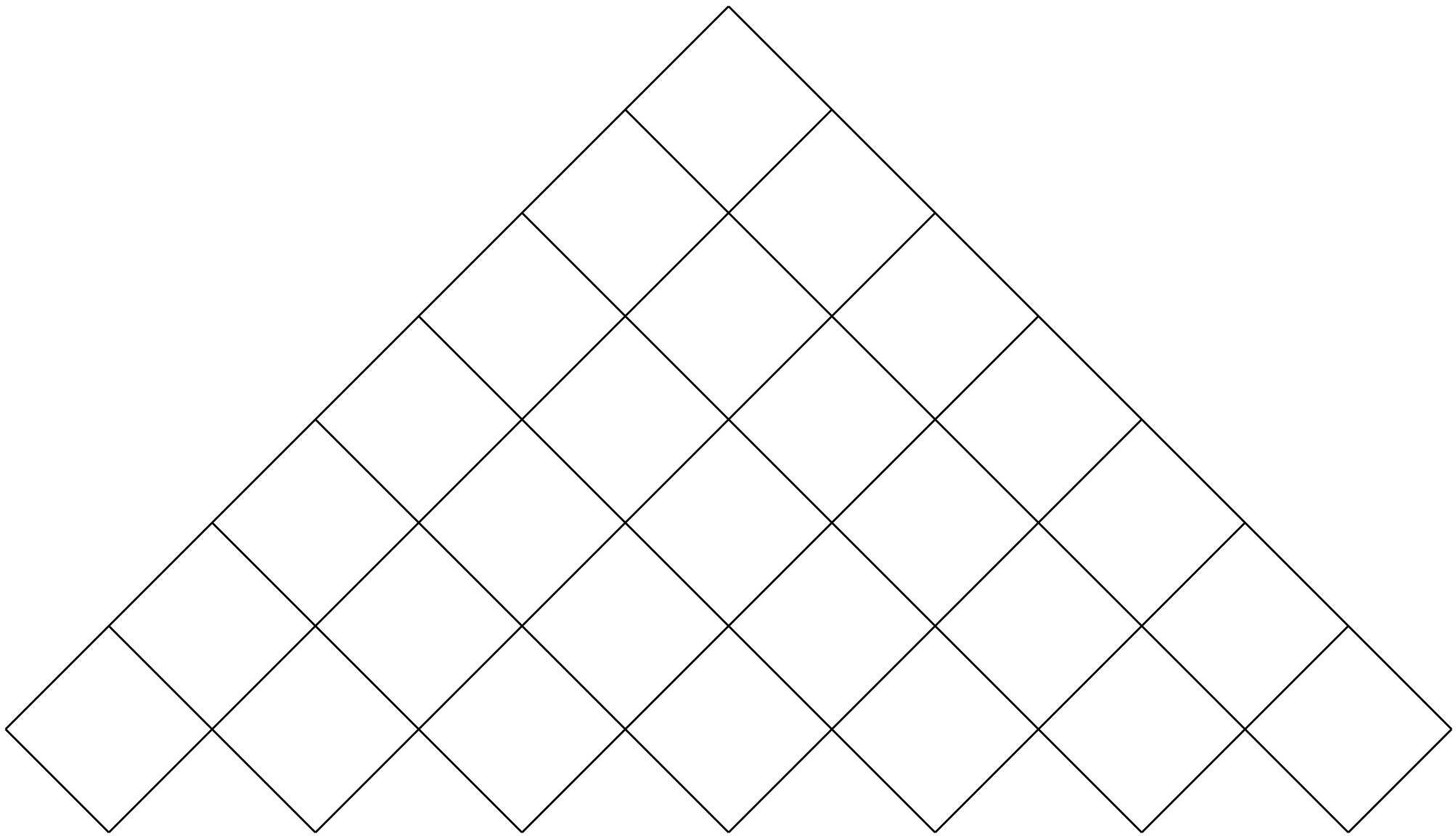
Data Structure

(Half a) two dimensional array (n x n)



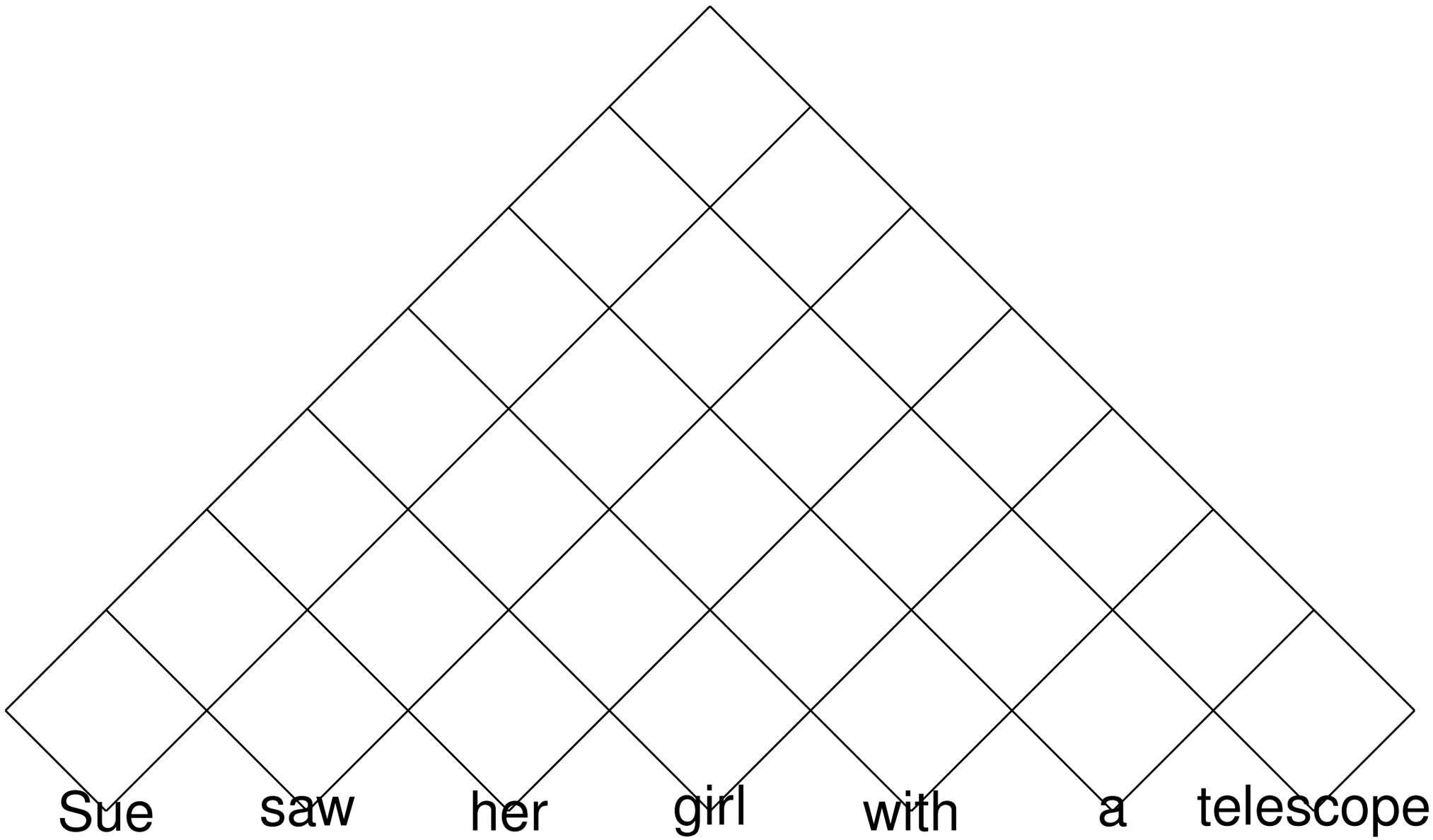
Data Structure

On its side



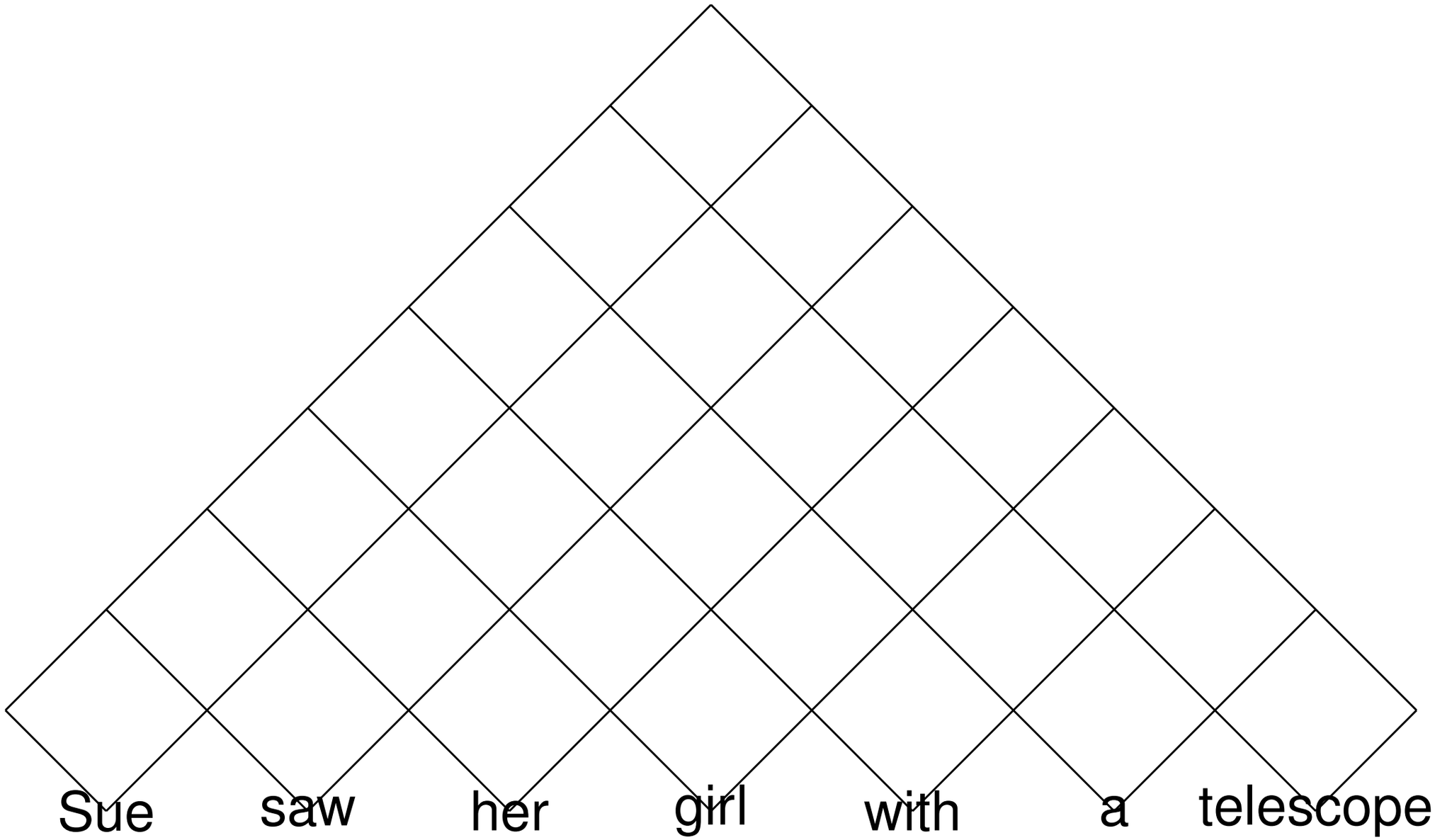
Data Structure

Each cell: all nonterminals than can derive word *i* to word *j*

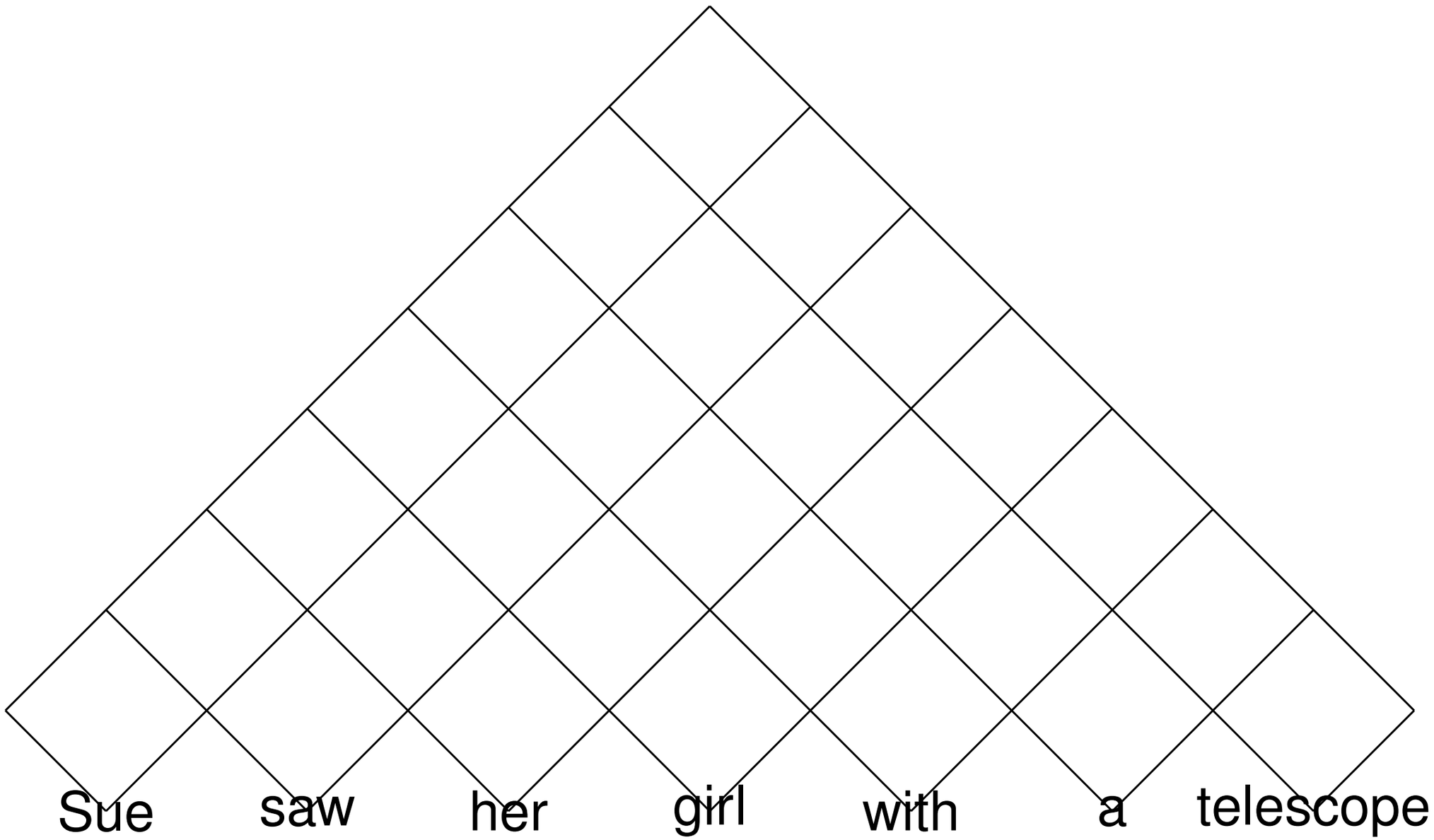


Data Structure

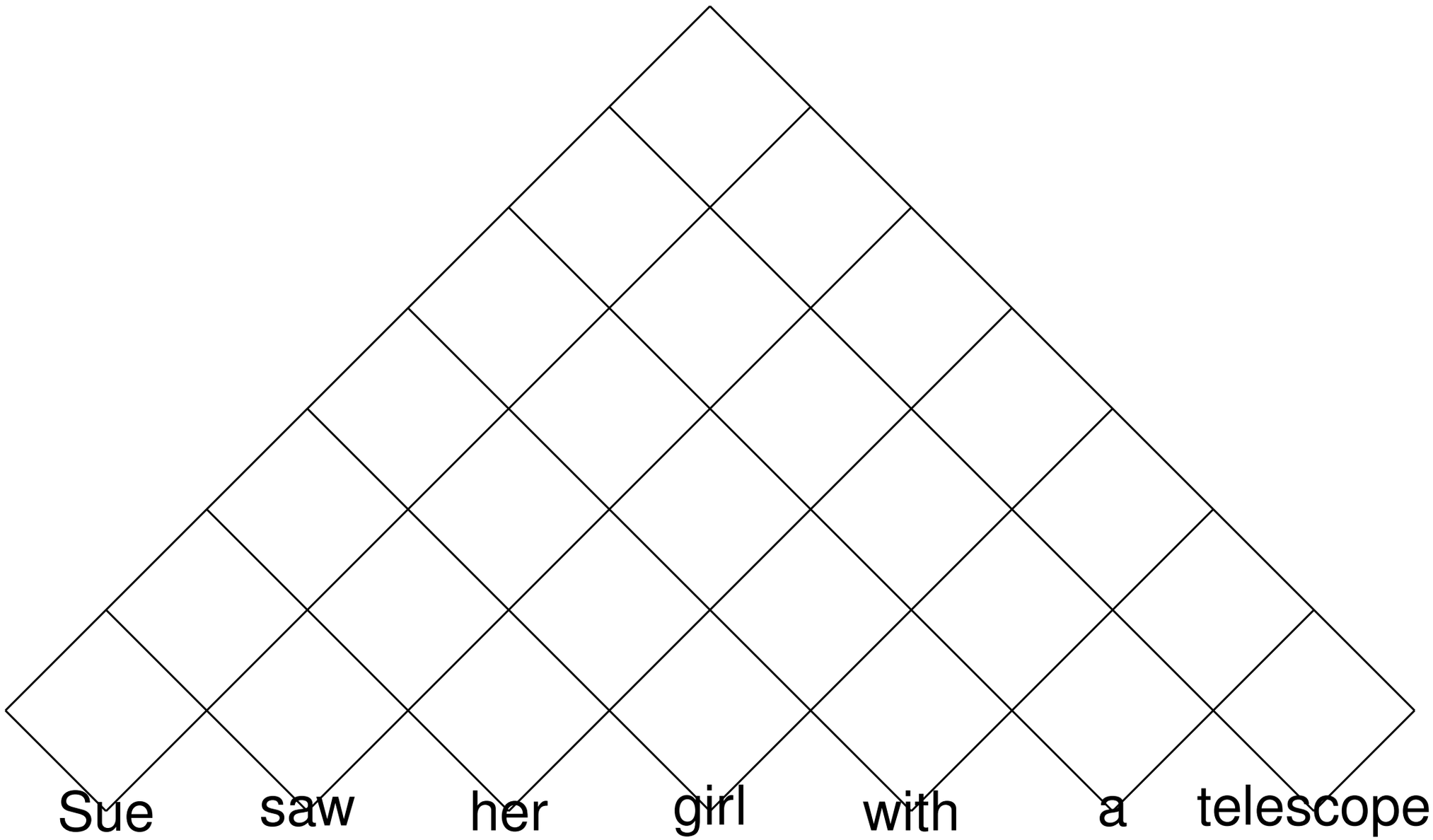
Each cell: all nonterminals than can derive word *i* to word *j*
imagine each cell as a *g* dimensional array



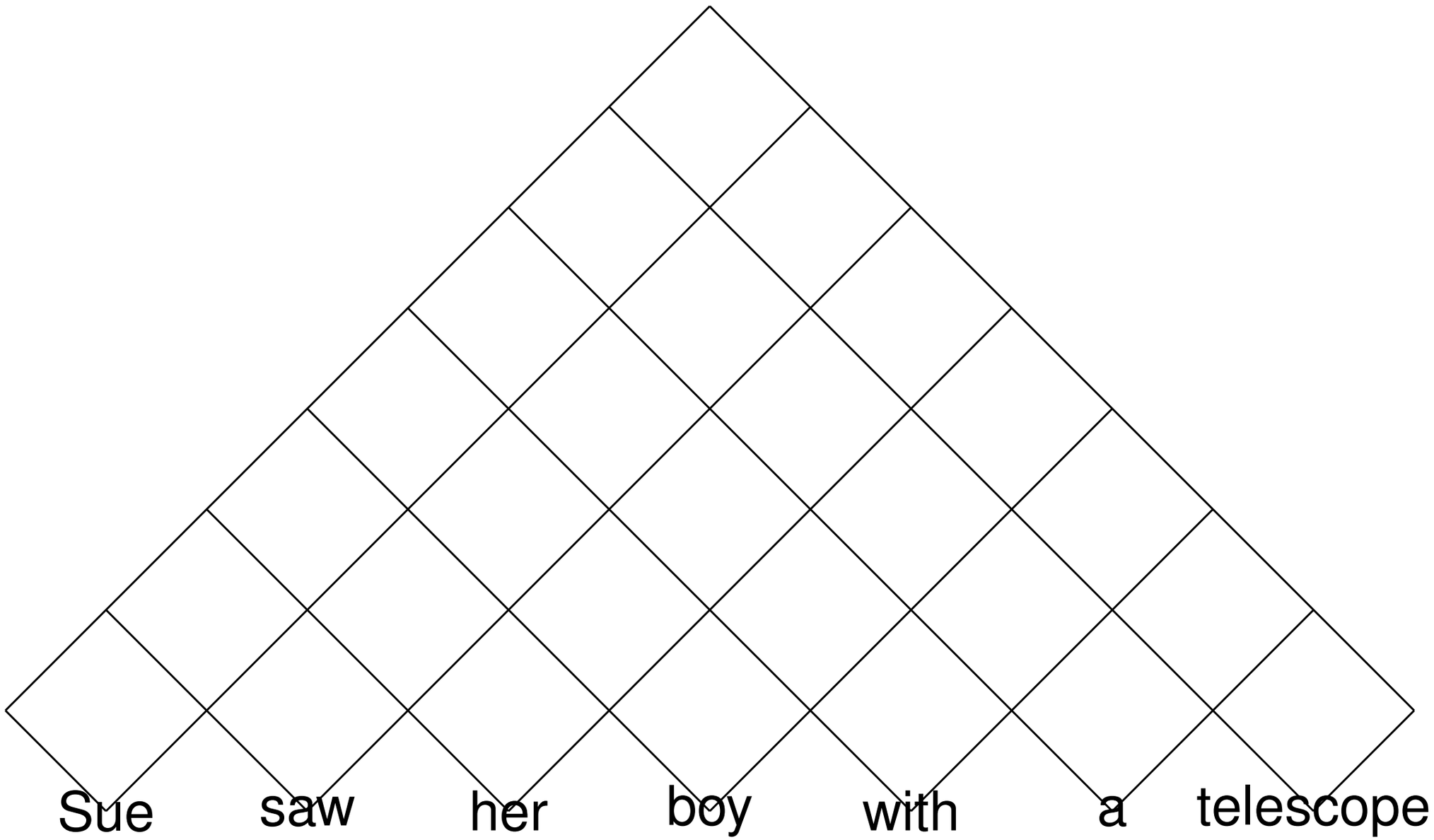
Filling the table



Handling Unary rules?



Which Order?



Complexity?

Complexity?

- $n^2 g$ cells to fill

Complexity?

- $n^2 g$ cells to fill
- $g^2 n$ ways to fill each one

Complexity?

- $n^2 g$ cells to fill
- $g^2 n$ ways to fill each one

$$O(g^3 n^3)$$

A Note on Implementation

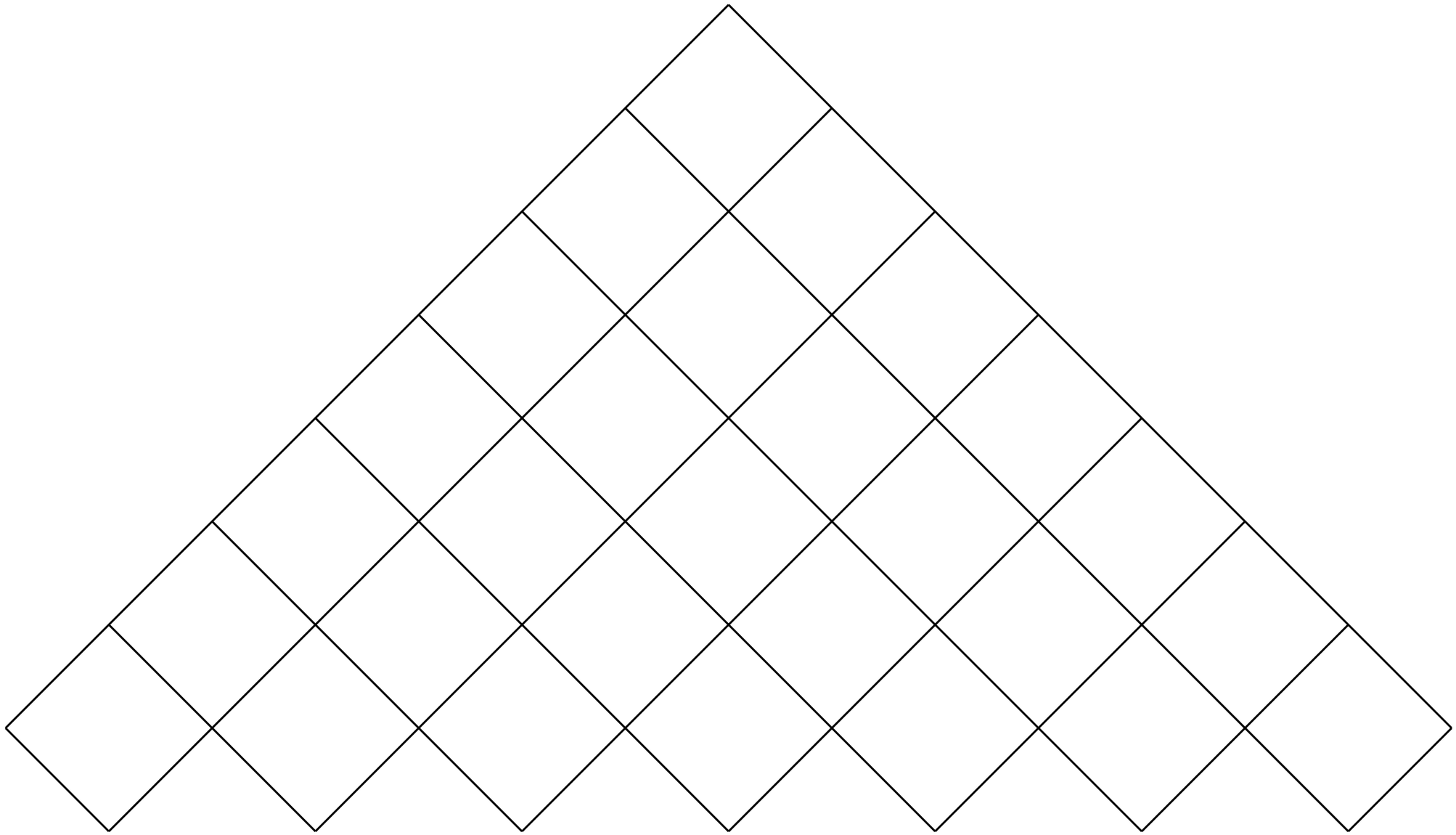
Smart implementation can reduce the runtime:

- Worst case is still $O(g^3 n^3)$, but it helps in practice
- No need to check all grammar rules $A \rightarrow BC$ at each location:
  - only those compatible with B or C of current split
  - prune binarized symbols which are too long for current position
  - once you found 1 way to derive A can break out of loop
  - order grammar rules from frequent to infrequent
- Need both efficient random access and iteration over possible symbols
  - Keep both hash and list, implemented as arrays

Finding a parse

Parsing – we want to actually find a parse tree

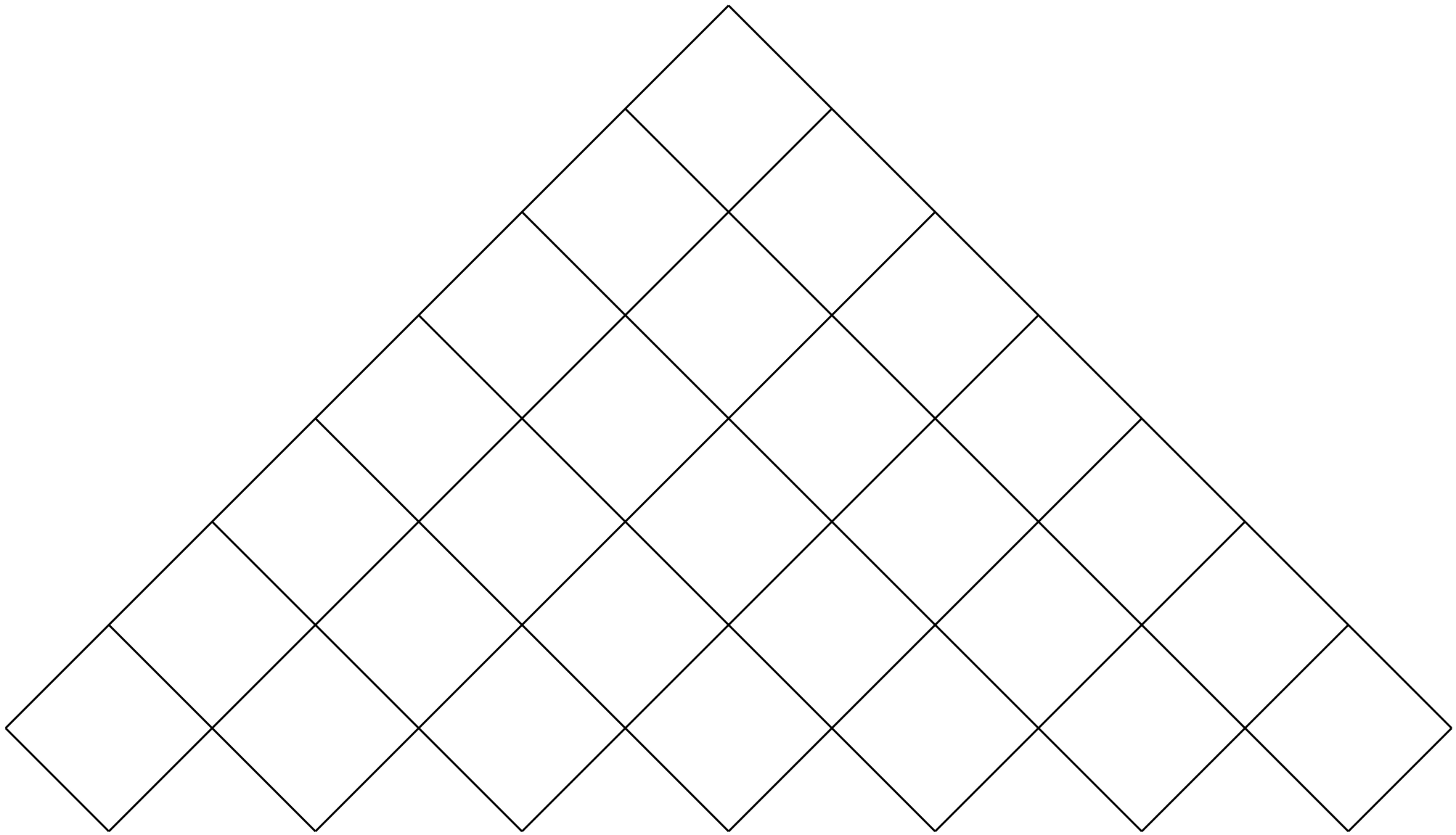
Easy: also keep a possible split point for each NT



PCFG Parsing and Disambiguation

Disambiguation – we want THE BEST parse tree

Easy: for each NT, keep best split point, and score.



Implementation Tricks

#1: sum instead of product

As in the HMM - Multiplying probabilities is evil

- keeping the product of many floating point numbers is dangerous, because product get really small
  - either grow in runtime
  - or loose precision (overflowing to 0)
  - either way, multiplying floats is expensive

Solution: use sum of logs instead

- remember: $log(p1 * p2) = log(p1) + log(p2)$

$\Rightarrow$ Use log probabilities instead of probabilities

$\Rightarrow$ add instead of multiply

Implementation Tricks

#2: Two passes

Some recognition speedup tricks are no longer possible

- need the best way to derive a symbol, not just one way
  - $\Rightarrow$ can't abort of loops early...

Solution: two passes

- pass 1: just recognition
- pass 2: disambiguation, with many pruned symbols at each cell

Summary

- Hierarchical Structure
- Constituent / Phrase-based Parsing
- CFGs, Derivations
- (Some) English Syntax
- PCFG
- CKY

Parsing with PCFG

- Parsing with a PCFG is finding the most probable derivation for a given sentence.
- This can be done quite efficiently with dynamic programming (the CKY algorithm)

Obtaining the probabilities

- We estimate them from the Treebank.
- $q(LHS \rightarrow RHS) = \frac{count(LHS \rightarrow RHS)}{count(LHS \rightarrow \Diamond)}$
- We can also add smoothing and backoff, as before.
- Dealing with unknown words - like in the HMM

The big question

Does this work?

Evaluation

Parsing Evaluation

- Let's assume we have a parser, how do we know how good it is?

$\Rightarrow$ Compare output trees to gold trees.

Parsing Evaluation

- Let's assume we have a parser, how do we know how good it is?

$\Rightarrow$ Compare output trees to gold trees.

  - But how do we compare trees?
  - Credit of 1 if tree is correct and 0 otherwise, is too harsh.

Parsing Evaluation

- Let's assume we have a parser, how do we know how good it is?

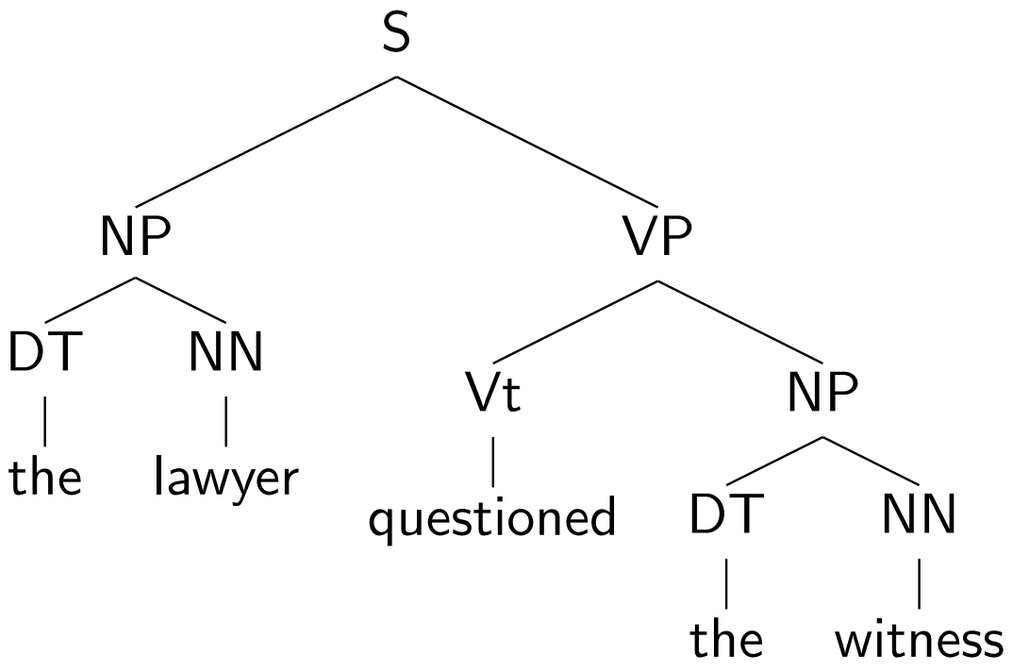
$\Rightarrow$ Compare output trees to gold trees.

  - But how do we compare trees?
  - Credit of 1 if tree is correct and 0 otherwise, is too harsh.

- Represent each tree as a set of labeled spans.
  - NP from word 1 to word 5.
  - VP from word 3 to word 4.
  - S from word 1 to word 23.
  - ...

- Measure Precision, Recall and F_1 over these spans, as in the segmentation case.

Evaluation: Representing Trees as Constituents



Label	Start Point	End Point
NP	1	2
NP	4	5
VP	3	5
S	1	5

(by Mike Collins)

Precision and Recall

Label	Start Point	End Point
NP	1	2
NP	4	5
NP	4	8
PP	6	8
NP	7	8
VP	3	8
S	1	8

Label	Start Point	End Point
NP	1	2
NP	4	5
PP	6	8
NP	7	8
VP	3	8
S	1	8

- $G =$ number of constituents in gold standard $= 7$
- $P =$ number in parse output $= 6$
- $C =$ number correct $= 6$

$$\text{Recall} = 100\% \times \frac{C}{G} = 100\% \times \frac{6}{7} \qquad \text{Precision} = 100\% \times \frac{C}{P} = 100\% \times \frac{6}{6}$$

(by Mike Collins)

Parsing Evaluation

- Is this a good measure?
  - Why? Why not?

(by Mike Collins)

Parsing Evaluation

How well does the PCFG parser we learned do?

Not very well: about 73% F_1 score.

(by Mike Collins)

Problems with PCFGs

(by Mike Collins)

Weaknesses of Probabilistic Context-Free Grammars

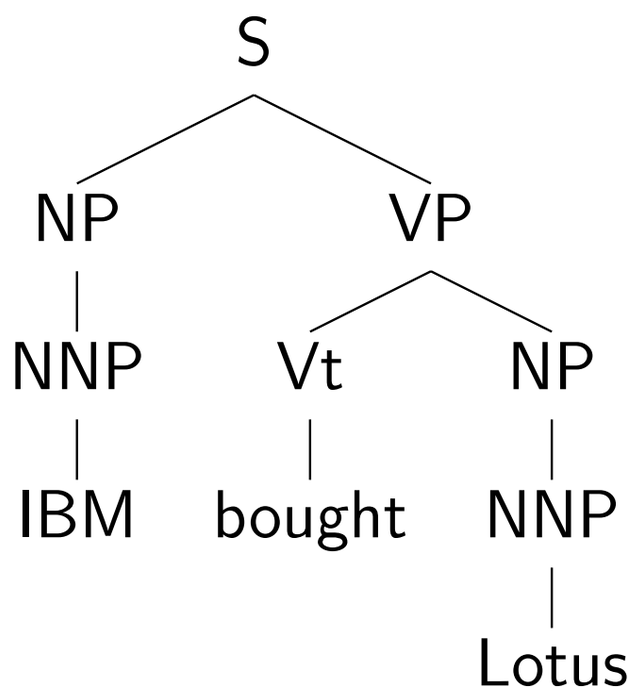
Michael Collins, Columbia University

(by Mike Collins)

Weaknesses of PCFGs

- Lack of sensitivity to lexical information
- Lack of sensitivity to structural frequencies

(by Mike Collins)

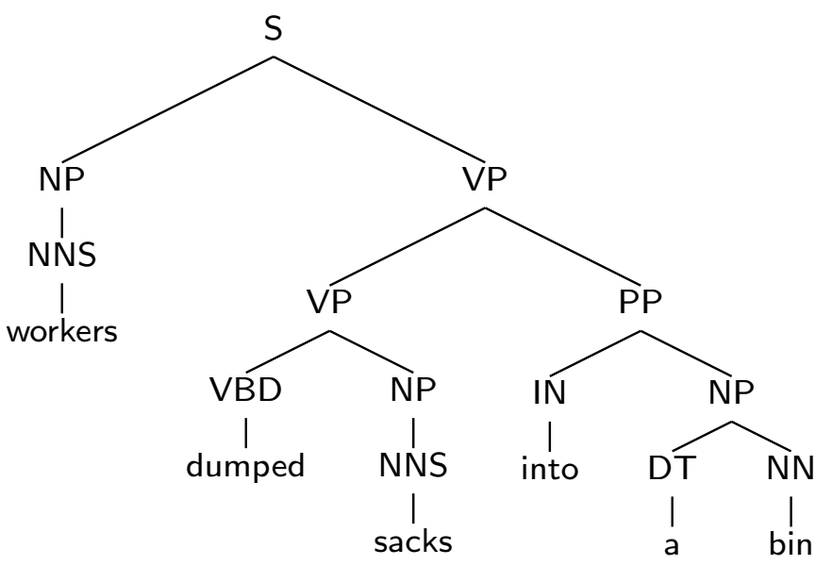


$$
\begin{aligned}
\mathsf{p(t)} = \quad & q(\mathsf{S} \rightarrow \mathsf{NP\ VP}) && \times q(\mathsf{NNP} \rightarrow IBM) \\
& \times q(\mathsf{VP} \rightarrow \mathsf{V\ NP}) && \times q(\mathsf{Vt} \rightarrow bought) \\
& \times q(\mathsf{NP} \rightarrow \mathsf{NNP}) && \times q(\mathsf{NNP} \rightarrow Lotus) \\
& \times q(\mathsf{NP} \rightarrow \mathsf{NNP})
\end{aligned}
$$

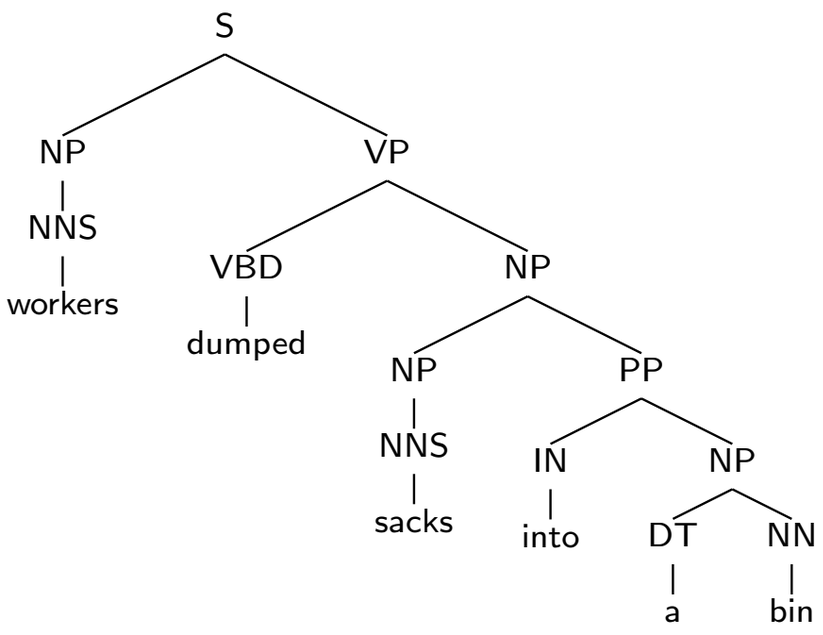
(by Mike Collins)

Another Case of PP Attachment Ambiguity

(a)



(b)



(by Mike Collins)

(a)

Rules
S $\rightarrow$ NP VP
NP $\rightarrow$ NNS
VP $\rightarrow$ VP PP
VP $\rightarrow$ VBD NP
NP $\rightarrow$ NNS
PP $\rightarrow$ IN NP
NP $\rightarrow$ DT NN
NNS $\rightarrow$ workers
VBD $\rightarrow$ dumped
NNS $\rightarrow$ sacks
IN $\rightarrow$ into
DT $\rightarrow$ a
NN $\rightarrow$ bin

(b)

Rules
S $\rightarrow$ NP VP
NP $\rightarrow$ NNS
NP $\rightarrow$ NP PP
VP $\rightarrow$ VBD NP
NP $\rightarrow$ NNS
PP $\rightarrow$ IN NP
NP $\rightarrow$ DT NN
NNS $\rightarrow$ workers
VBD $\rightarrow$ dumped
NNS $\rightarrow$ sacks
IN $\rightarrow$ into
DT $\rightarrow$ a
NN $\rightarrow$ bin

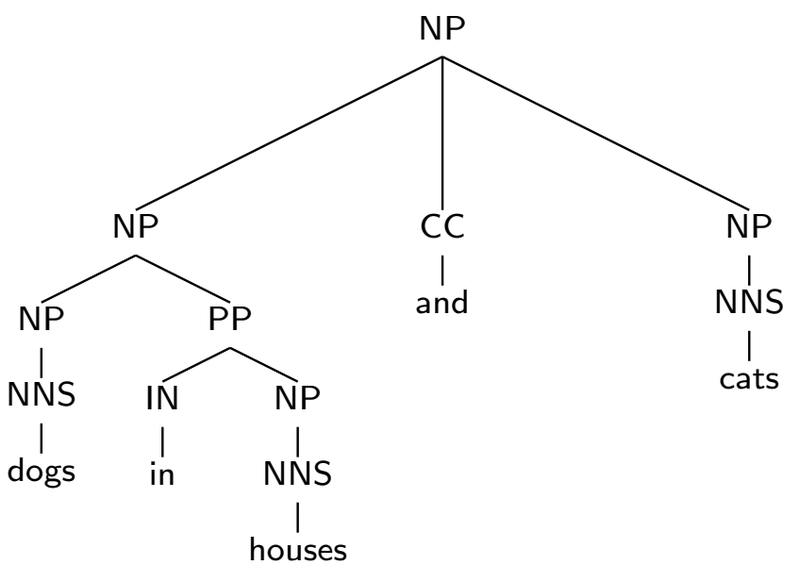
If $q(\text{NP} \rightarrow \text{NP PP}) > q(\text{VP} \rightarrow \text{VP PP})$ then (b) is more probable, else (a) is more probable.

Attachment decision is completely independent of the words

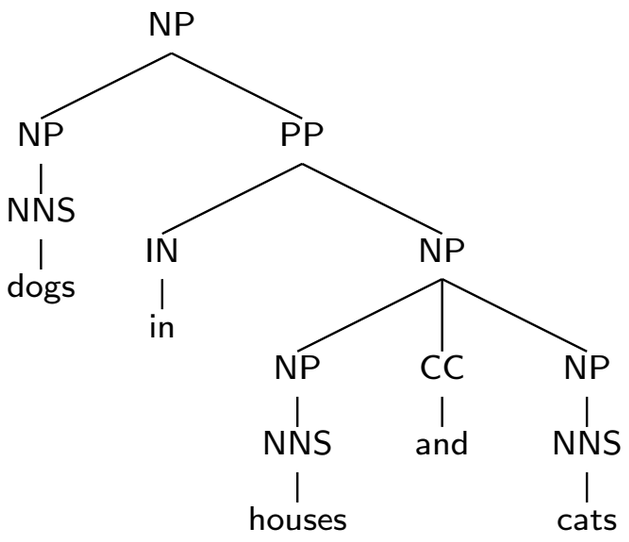
(by Mike Collins)

A Case of Coordination Ambiguity

(a)



(b)



(by Mike Collins)

(a)

Rules
NP $\rightarrow$ NP CC NP
NP $\rightarrow$ NP PP
NP $\rightarrow$ NNS
PP $\rightarrow$ IN NP
NP $\rightarrow$ NNS
NP $\rightarrow$ NNS
NNS $\rightarrow$ dogs
IN $\rightarrow$ in
NNS $\rightarrow$ houses
CC $\rightarrow$ and
NNS $\rightarrow$ cats

(b)

Rules
NP $\rightarrow$ NP CC NP
NP $\rightarrow$ NP PP
NP $\rightarrow$ NNS
PP $\rightarrow$ IN NP
NP $\rightarrow$ NNS
NP $\rightarrow$ NNS
NNS $\rightarrow$ dogs
IN $\rightarrow$ in
NNS $\rightarrow$ houses
CC $\rightarrow$ and
NNS $\rightarrow$ cats

Here the two parses have identical rules, and therefore have identical probability under any assignment of PCFG rule probabilities

(by Mike Collins)