COS 484: Natural Language Processing

Constituency Parsing

Fall 2019

(Some slides adapted from Chris Manning, Mike Collins)

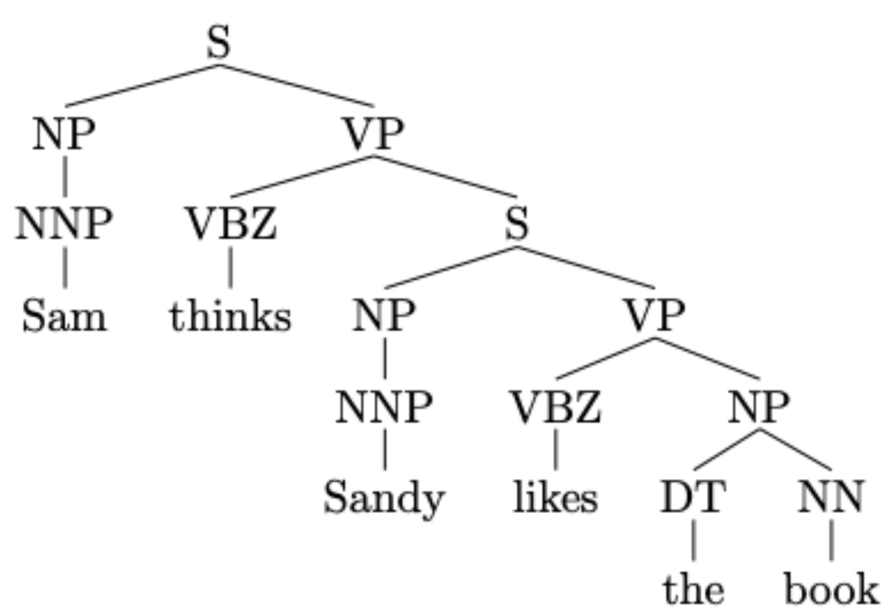
Overview

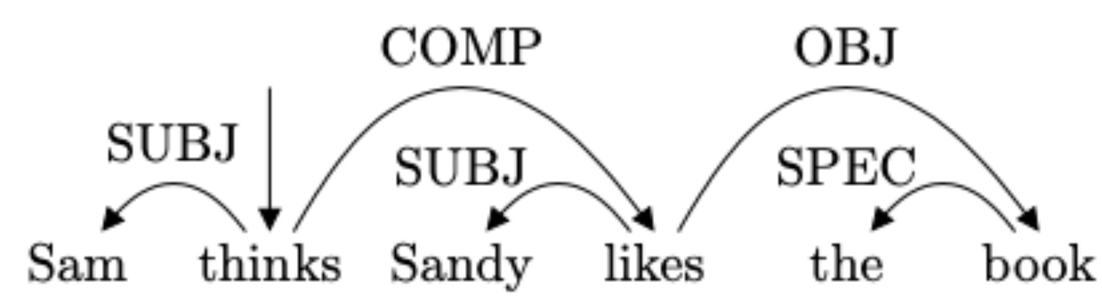
- Constituency structure vs dependency structure
- Context-free grammar (CFG)
- Probabilistic context-free grammar (PCFG)
- The CKY algorithm
- Evaluation
- Lexicalized PCFGs

Syntactic structure: constituency and dependency

Two views of linguistic structure

- **Constituency**
  - = phrase structure grammar
  - = context-free grammars (CFGs)
- **Dependency**





Constituency structure

- **Phrase structure** organizes words into **nested constituents**
- Starting units: words are given a category: part-of-speech tags

the, cuddly, cat, by, the, door

Det, Adj, N, P, Det, N

- Words combine into phrases with categories

the cuddly cat, by the door

NP→Det Adj N   PP→P NP

- Phrases can combine into bigger phrases recursively

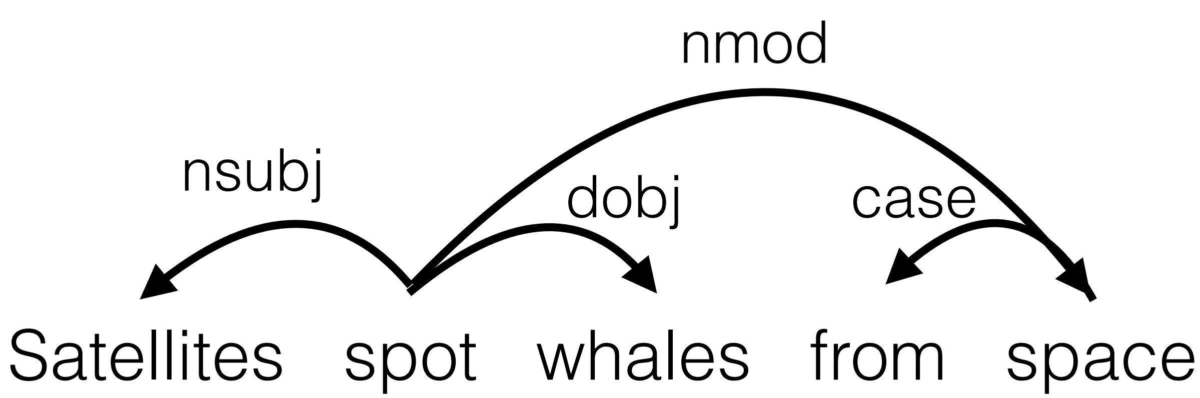
the cuddly cat by the door

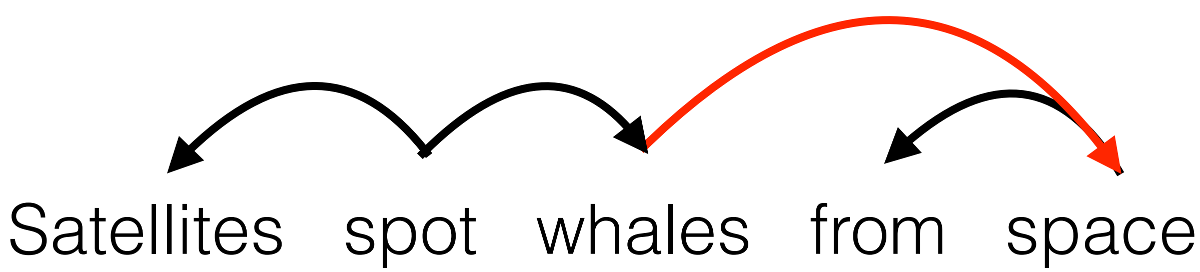
NP→ NP PP

This Thursday

Dependency structure

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





Why do we need sentence structure?

- We need to understand sentence structure in order to be able to interpret language correctly

- Human communicate complex ideas by composing words together into bigger units

- We need to know what is connected to what

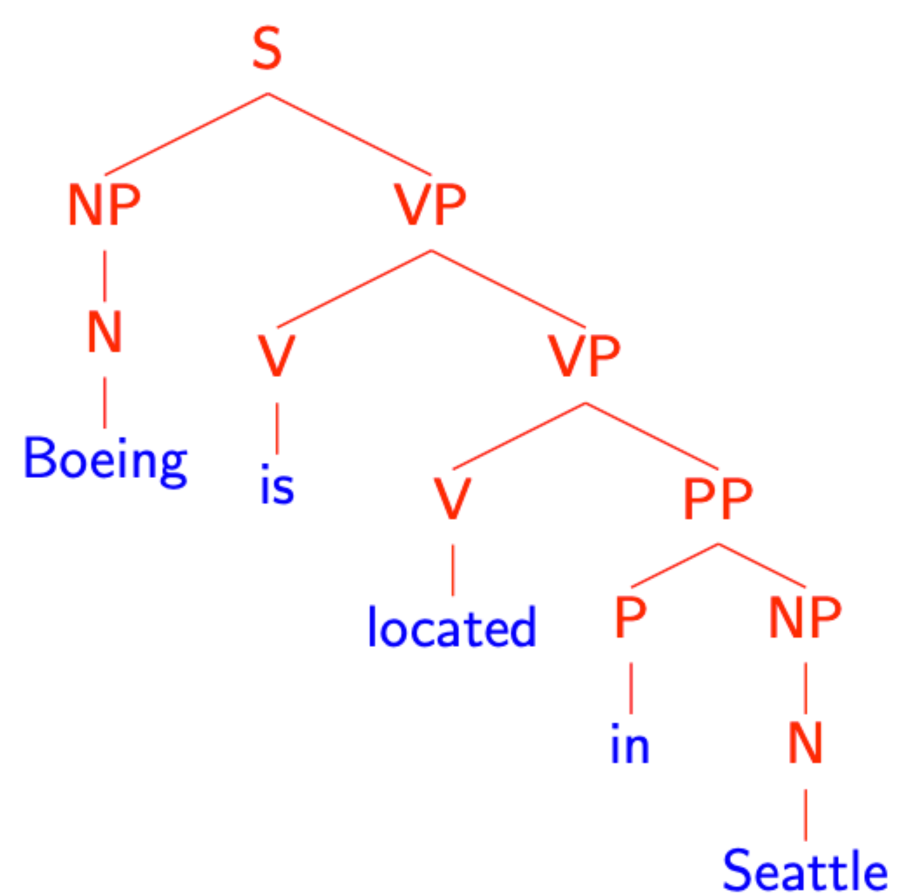
Syntactic parsing

- Syntactic parsing is the task of recognizing a sentence and assigning a structure to it.

Input:

Beoing is located in Seattle.

Output:



Syntactic parsing

- Used as intermediate representation for downstream applications

English word order: subject — verb — object

Japanese word order: subject — object — verb

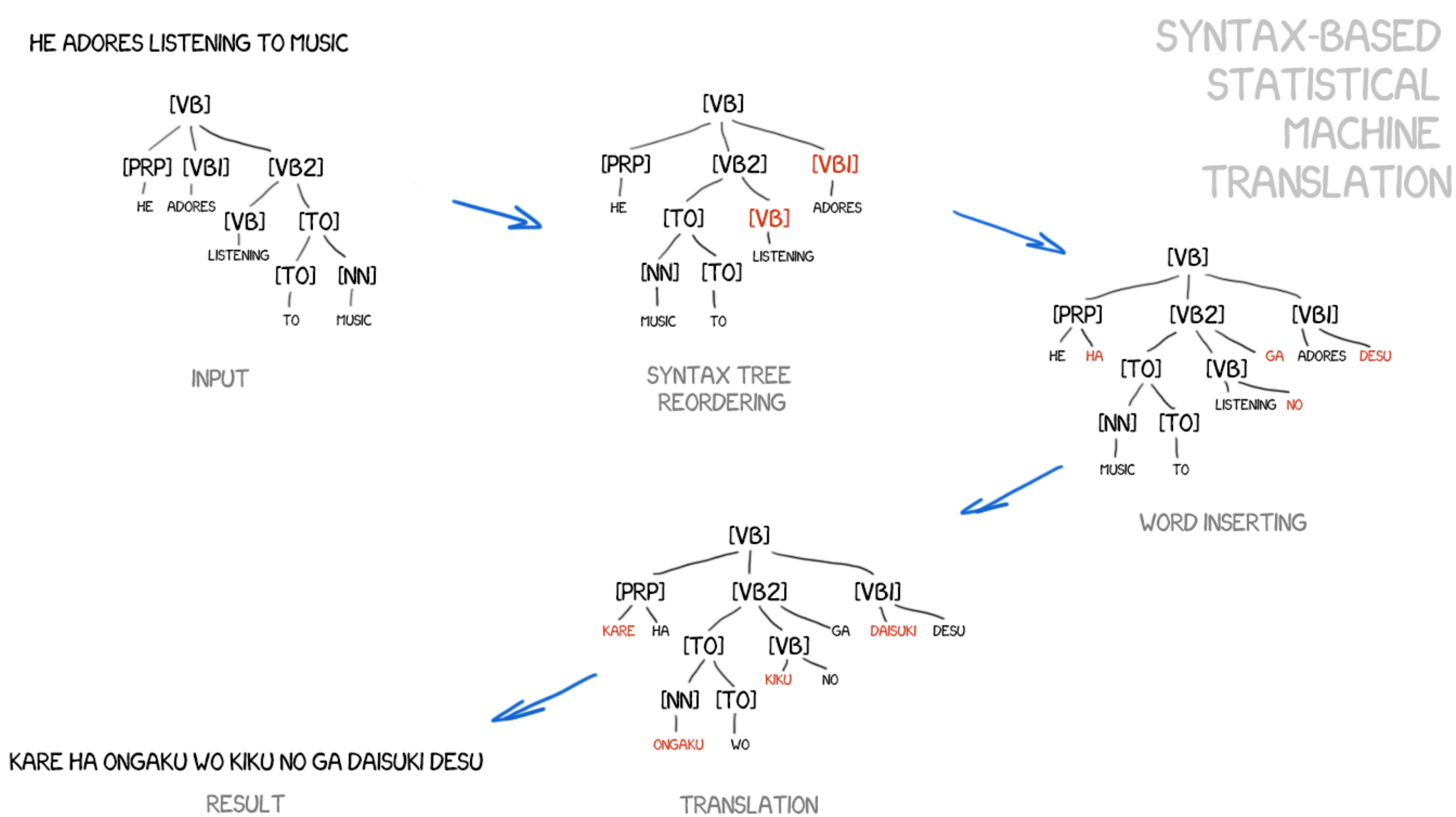


Image credit: http://vas3k.com/blog/machine_translation/

Syntactic parsing

- Used as intermediate representation for downstream applications

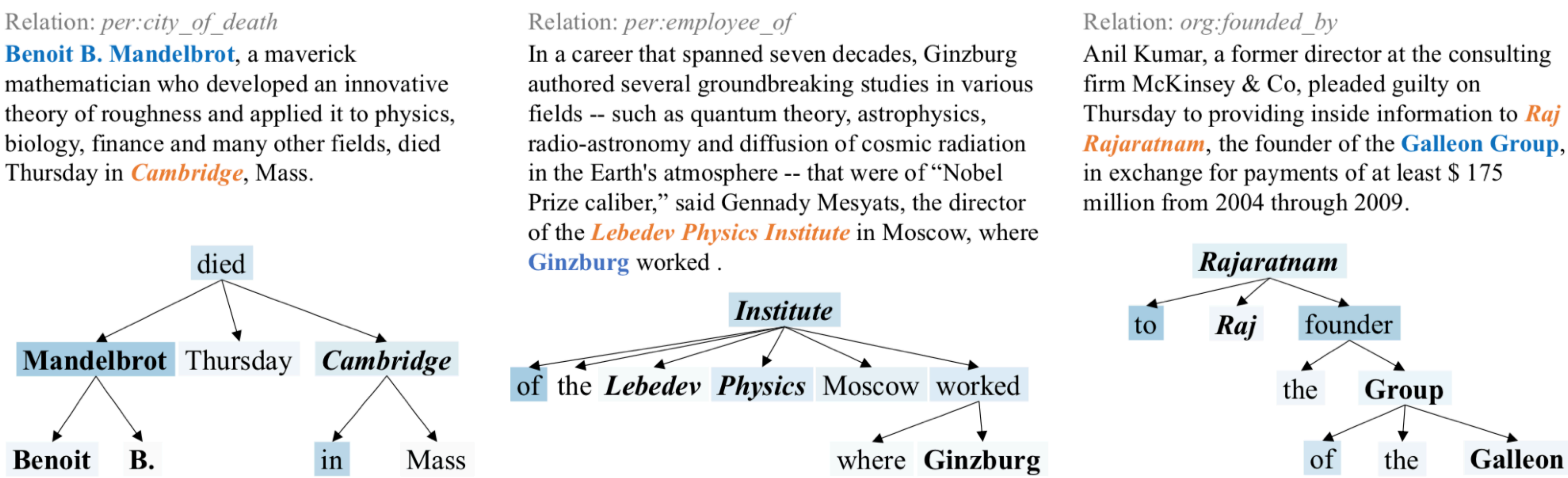
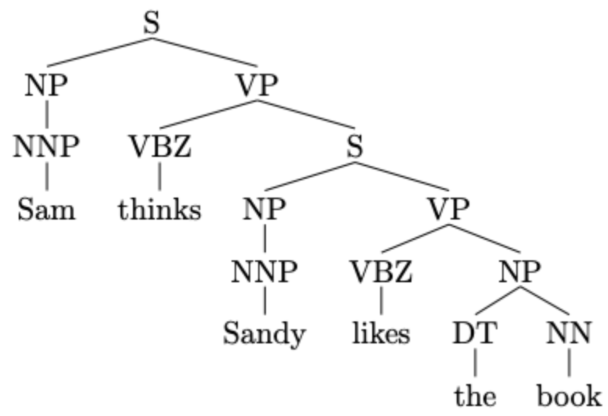


Image credit: (Zhang et al, 2018)



Context-free grammars

- The most widely used formal system for modeling constituency structure in English and other natural languages
- 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 $X \rightarrow Y_1 Y_2 \ldots Y_n$ for $n \geq 1$, $X \in N, Y_i \in (N \cup \Sigma)$
  - $S \in N$ is a distinguished start symbol

A Context-Free Grammar for English

N = {S, NP, VP, PP, DT, Vi, Vt, NN, IN}
S = S
Σ = {sleeps, saw, man, woman, telescope, the, with, in}

$R =$

S	$\rightarrow$	NP	VP
VP	$\rightarrow$	Vi	
VP	$\rightarrow$	Vt	NP
VP	$\rightarrow$	VP	PP
NP	$\rightarrow$	DT	NN
NP	$\rightarrow$	NP	PP
PP	$\rightarrow$	IN	NP

Grammar

Vi	$\rightarrow$	sleeps
Vt	$\rightarrow$	saw
NN	$\rightarrow$	man
NN	$\rightarrow$	woman
NN	$\rightarrow$	telescope
NN	$\rightarrow$	dog
DT	$\rightarrow$	the
IN	$\rightarrow$	with
IN	$\rightarrow$	in

Lexicon

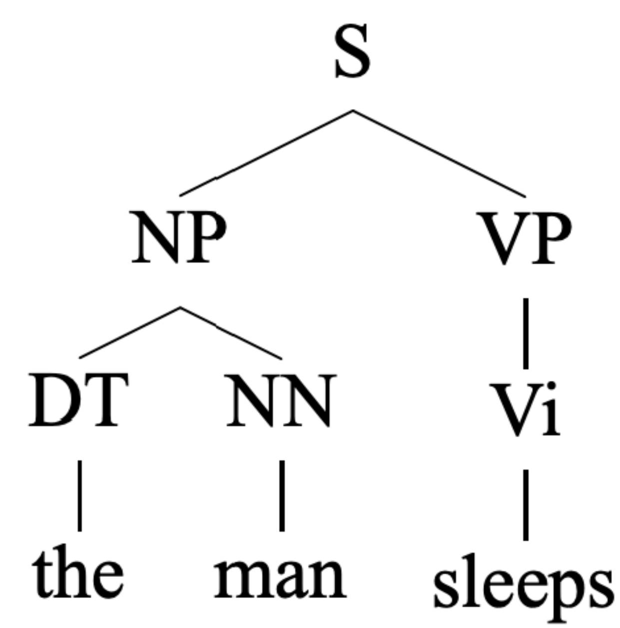
S:sentence, VP:verb phrase, NP: noun phrase, PP:prepositional phrase, DT:determiner, Vi:intransitive verb, Vt:transitive verb, NN: noun, IN:preposition

(Left-most) Derivations

- Given a CFG G, a left-most derivation is a sequence of strings $s_1, s_2, \ldots, s_n$, where
  - $s_1 = S$
  - $s_n \in \Sigma^*$: all possible strings made up of words from Σ
  - Each s_i for $i = 2, \ldots, n$ 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 $X \to \beta \in R$
- s_n: yield of the derivation

(Left-most) Derivations

- $s_1 = \text{S}$
- $s_2 = \text{NP VP}$
- $s_3 = \text{DT NN VP}$
- $s_4 = \text{the NN VP}$
- $s_5 = \text{the man VP}$
- $s_6 = \text{the man Vi}$
- $s_7 = \text{the man sleeps}$



A derivation can be represented as a parse tree!

$R =$

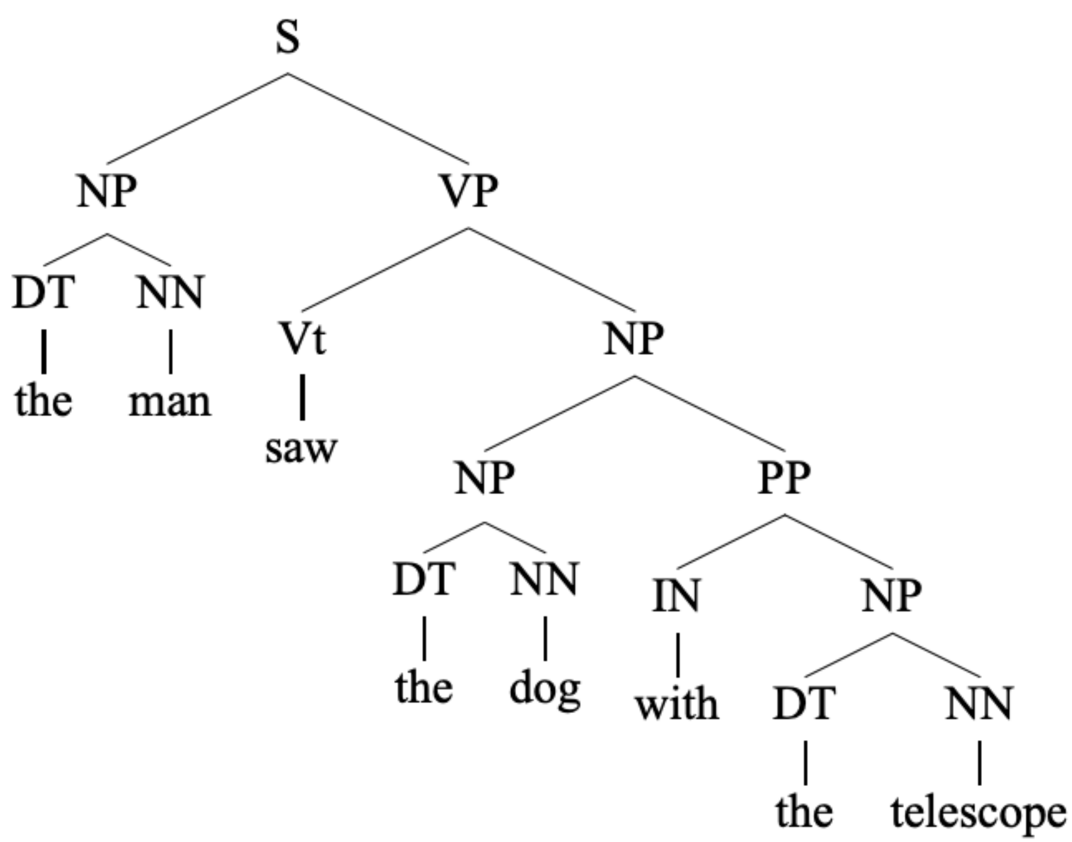
S	$\rightarrow$	NP	VP
VP	$\rightarrow$	Vi	
VP	$\rightarrow$	Vt	NP
VP	$\rightarrow$	VP	PP
NP	$\rightarrow$	DT	NN
NP	$\rightarrow$	NP	PP
PP	$\rightarrow$	IN	NP

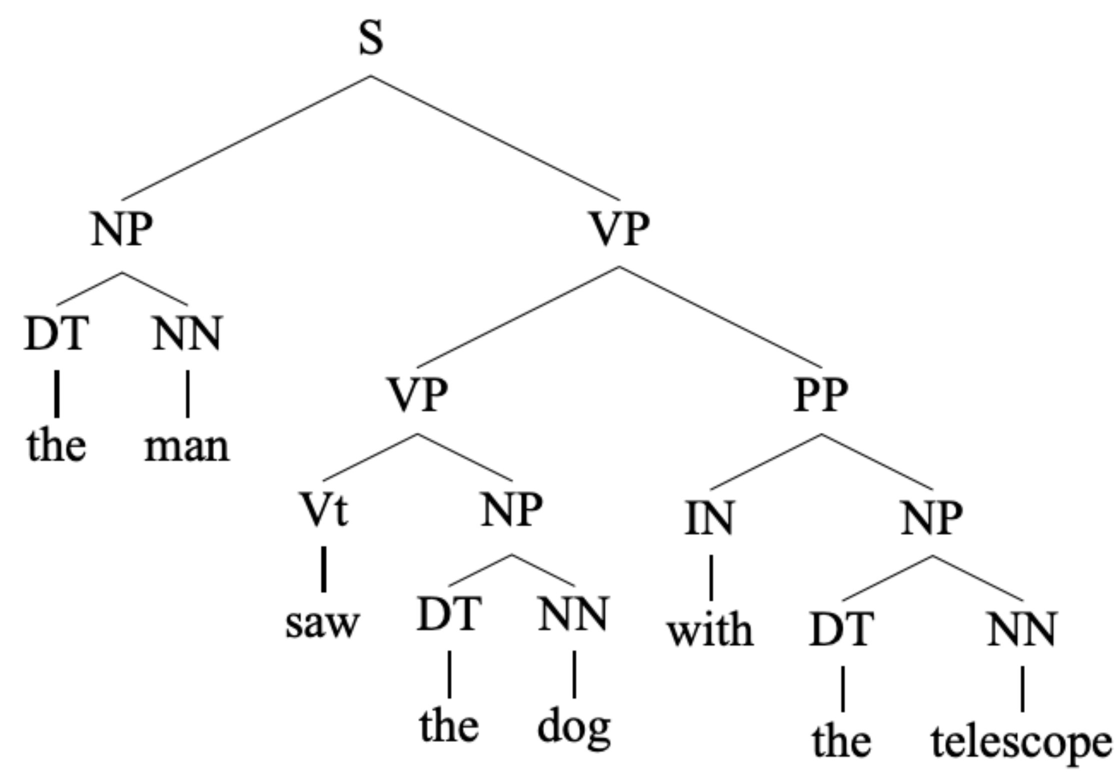
Vi	$\rightarrow$	sleeps
Vt	$\rightarrow$	saw
NN	$\rightarrow$	man
NN	$\rightarrow$	woman
NN	$\rightarrow$	telescope
NN	$\rightarrow$	dog
DT	$\rightarrow$	the
IN	$\rightarrow$	with
IN	$\rightarrow$	in

- A string $s \in \Sigma^*$ is in the language defined by the CFG if there is at least one derivation whose yield is s
- The set of possible derivations may be finite or infinite

Ambiguity

- Some strings may have more than one derivations (i.e. more than one parse trees!).





“Classical” NLP Parsing

- In fact, sentences can have a very large number of possible parses

  The board approved [its acquisition] [by Royal Trustco Ltd.] [of Toronto] [for $27 a share] [at its monthly meeting].

((ab)c)d (a(bc))d (ab)(cd) a((bc)d) a(b(cd))

Catalan number: $C_n = \frac{1}{n+1}\binom{2n}{n}$

- It is also difficult to construct a grammar with enough coverage
  - A less constrained grammar can parse more sentences but result in more parses for even simple sentences
  - There is no way to choose the right parse!

Statistical parsing

- **Learning from data**: treebanks
- **Adding probabilities to the rules**: probabilistic CFGs (PCFGs)

Treebanks: a collection of sentences paired with their parse trees

```
((S
   (NP-SBJ (DT That)
     (JJ cold) (, ,)
     (JJ empty) (NN sky) )
   (VP (VBD was)
     (ADJP-PRD (JJ full)
       (PP (IN of)
         (NP (NN fire)
           (CC and)
           (NN light) ))))
   (. .) ))
```

(a)

```
((S
   (NP-SBJ The/DT flight/NN )
   (VP should/MD
     (VP arrive/VB
       (PP-TMP at/IN
         (NP eleven/CD a.m/RB ))
       (NP-TMP tomorrow/NN )))))
```

(b)

The Penn Treebank Project (Marcus et al, 1993)

Treebanks

- Standard setup (WSJ portion of Penn Treebank):
  - 40,000 sentences for training
  - 1,700 for development
  - 2,400 for testing

- Why building a treebank instead of a grammar?
  - Broad coverage
  - Frequencies and distributional information
  - A way to evaluate systems

Probabilistic context-free grammars (PCFGs)

S	$\Rightarrow$	NP	VP	1.0
VP	$\Rightarrow$	Vi		0.4
VP	$\Rightarrow$	Vt	NP	0.4
VP	$\Rightarrow$	VP	PP	0.2
NP	$\Rightarrow$	DT	NN	0.3
NP	$\Rightarrow$	NP	PP	0.7
PP	$\Rightarrow$	P	NP	1.0

Vi	$\Rightarrow$	sleeps	1.0
Vt	$\Rightarrow$	saw	1.0
NN	$\Rightarrow$	man	0.7
NN	$\Rightarrow$	woman	0.2
NN	$\Rightarrow$	telescope	0.1
DT	$\Rightarrow$	the	1.0
IN	$\Rightarrow$	with	0.5
IN	$\Rightarrow$	in	0.5

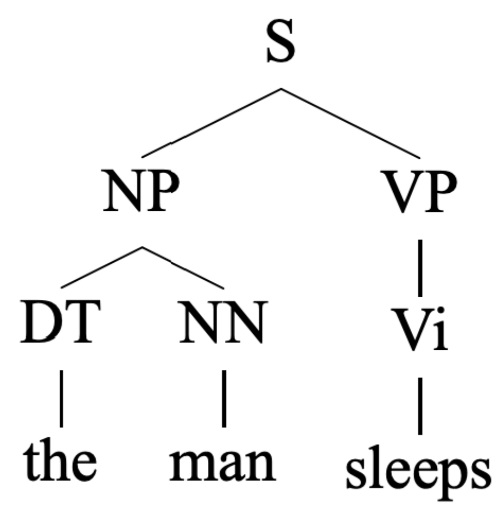
- A probabilistic context-free grammar (PCFG) consists of:
  - A context-free grammar: $G = (N, \Sigma, R, S)$
  - For each rule $\alpha \to \beta \in R$, there is a parameter $q(\alpha \to \beta) \geq 0$. For any $X \in N$,

$$\sum_{\alpha\to\beta:\alpha=X} q(\alpha \to \beta) = 1$$

Probabilistic context-free grammars (PCFGs)

For any derivation (parse tree) containing rules:
$\alpha_1 \to \beta_1, \alpha_2 \to \beta_2, \ldots, \alpha_l \to \beta_l$, the probability of the parse is:

$$\prod_{i=1}^{l} q(\alpha_i \to \beta_i)$$



S	$\Rightarrow$	NP	VP	1.0
VP	$\Rightarrow$	Vi		0.4
VP	$\Rightarrow$	Vt	NP	0.4
VP	$\Rightarrow$	VP	PP	0.2
NP	$\Rightarrow$	DT	NN	0.3
NP	$\Rightarrow$	NP	PP	0.7
PP	$\Rightarrow$	P	NP	1.0

Vi	$\Rightarrow$	sleeps	1.0
Vt	$\Rightarrow$	saw	1.0
NN	$\Rightarrow$	man	0.7
NN	$\Rightarrow$	woman	0.2
NN	$\Rightarrow$	telescope	0.1
DT	$\Rightarrow$	the	1.0
IN	$\Rightarrow$	with	0.5
IN	$\Rightarrow$	in	0.5

$$P(t) = q(\mathrm{S} \to \mathrm{NP\ VP}) \times q(\mathrm{NP} \to \mathrm{DT\ NN}) \times q(\mathrm{DT} \to \mathrm{the}) \times q(\mathrm{NN} \to \mathrm{man}) \times q(\mathrm{VP} \to \mathrm{Vi}) \times q(\mathrm{Vi} \to \mathrm{sleeps})$$

$$= 1.0 \times 0.3 \times 1.0 \times 0.7 \times 0.4 \times 1.0 = 0.084$$

Why do we want $\displaystyle\sum_{\alpha\to\beta:\alpha=X} q(\alpha \to \beta) = 1$?

Deriving a PCFG from a treebank

- Training data: a set of parse trees $t_1, t_2, \ldots, t_m$
- A PCFG (N, Σ, S, R, q):
  - N is the set of all non-terminals seen in the trees
  - Σ is the set of all words seen in the trees
  - S is taken to be S.
  - R is taken to be the set of all rules $\alpha \to \beta$ seen in the trees
  - The maximum-likelihood parameter estimates are:

$$q_{ML}(\alpha \to \beta) = \frac{\text{Count}(\alpha \to \beta)}{\text{Count}(\alpha)}$$

If we have seen the rule VP $\to$ Vt NP 105 times, and the the non-terminal VP 1000 times, $q(\text{VP} \to \text{Vt NP}) = 0.105$

Parsing with PCFGs

- Given a sentence s and a PCFG, how to find the highest scoring parse tree for s?

$$argmax_{t \in \mathcal{T}(s)} P(t)$$

- **The CKY algorithm**: applies to a PCFG in Chomsky normal form (CNF)

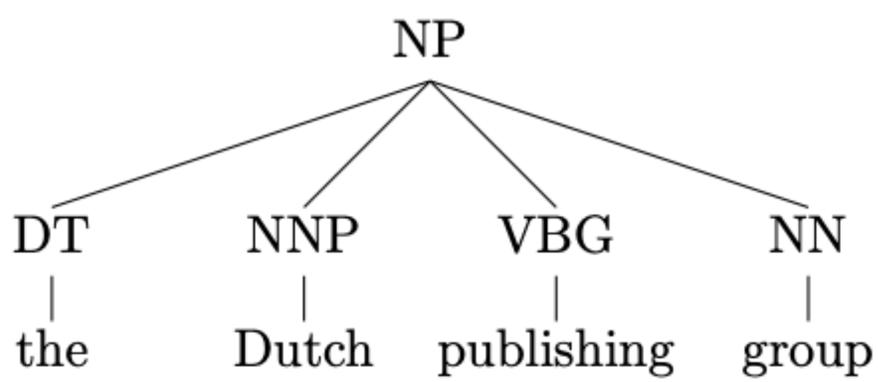
- **Chomsky Normal Form (CNF)**: all the rules take one of the two following forms:

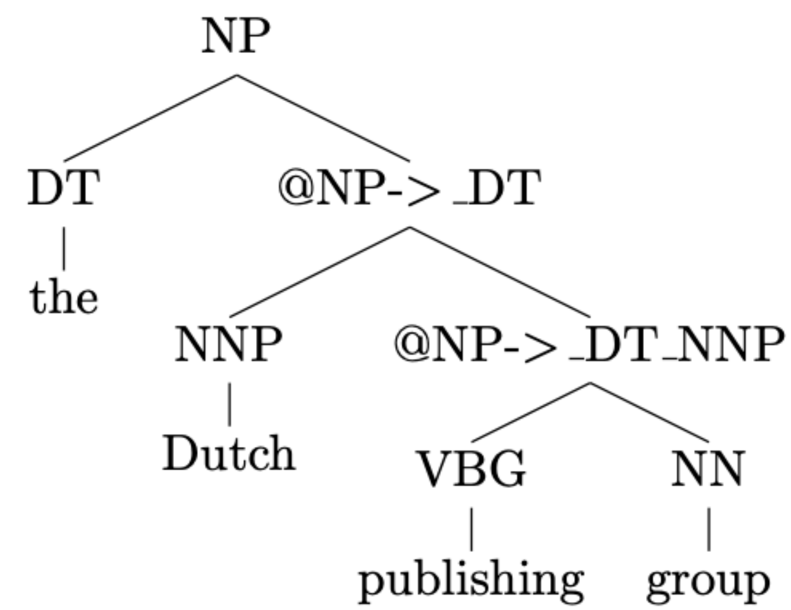
    - $X \rightarrow Y_1 Y_2$ where $X \in N, Y_1 \in N, Y_2 \in N$
    - $X \rightarrow Y$ where $X \in N, Y \in \Sigma$

- It is possible to convert any PCFG into an equivalent grammar in CNF!
    - However, the trees will look differently; It is possible to do "reverse transformation"

Converting PCFGs into a CNF grammar

- n-ary rules ($n > 2$): NP → DT NNP VBG NN



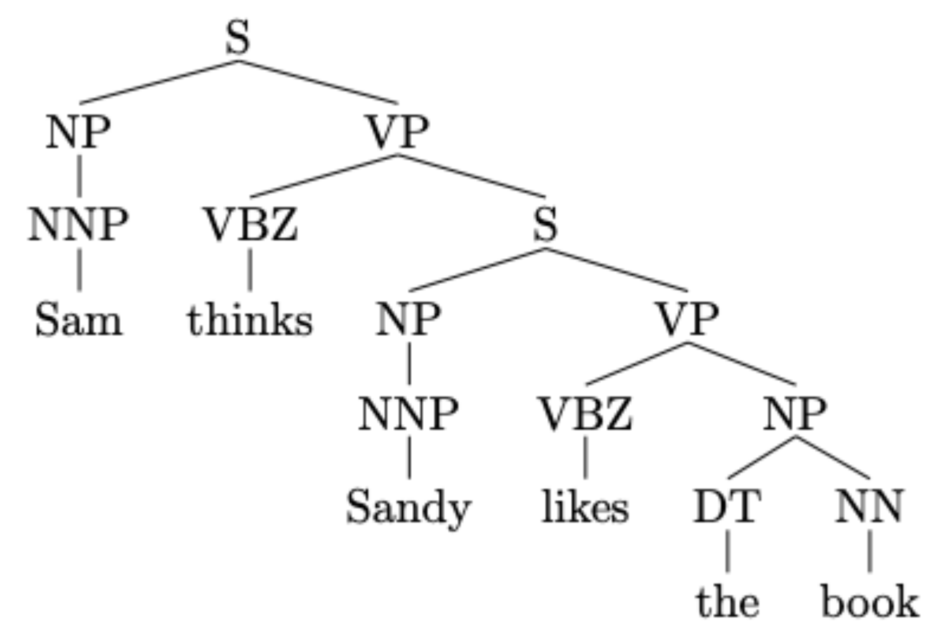


- Unary rules: VP → Vi, Vi → sleeps
  - Eliminate all the unary rules recursively by adding VP → sleeps
  - We will come back to this later!

The CKY algorithm

- Dynamic programming
- Given a sentence $x_1, x_2, \ldots, x_n$, denote $\pi(i, j, X)$ as the highest score for any parse tree that dominates words $x_i, \ldots, x_j$ and has non-terminal $X \in N$ as its root.
- Output: $\pi(1, n, S)$
- Initially, for $i = 1, 2, \ldots, n$,

$$\pi(i, i, X) = \begin{cases} q(X \to x_i) & \text{if } X \to x_i \in R \\ 0 & \text{otherwise} \end{cases}$$

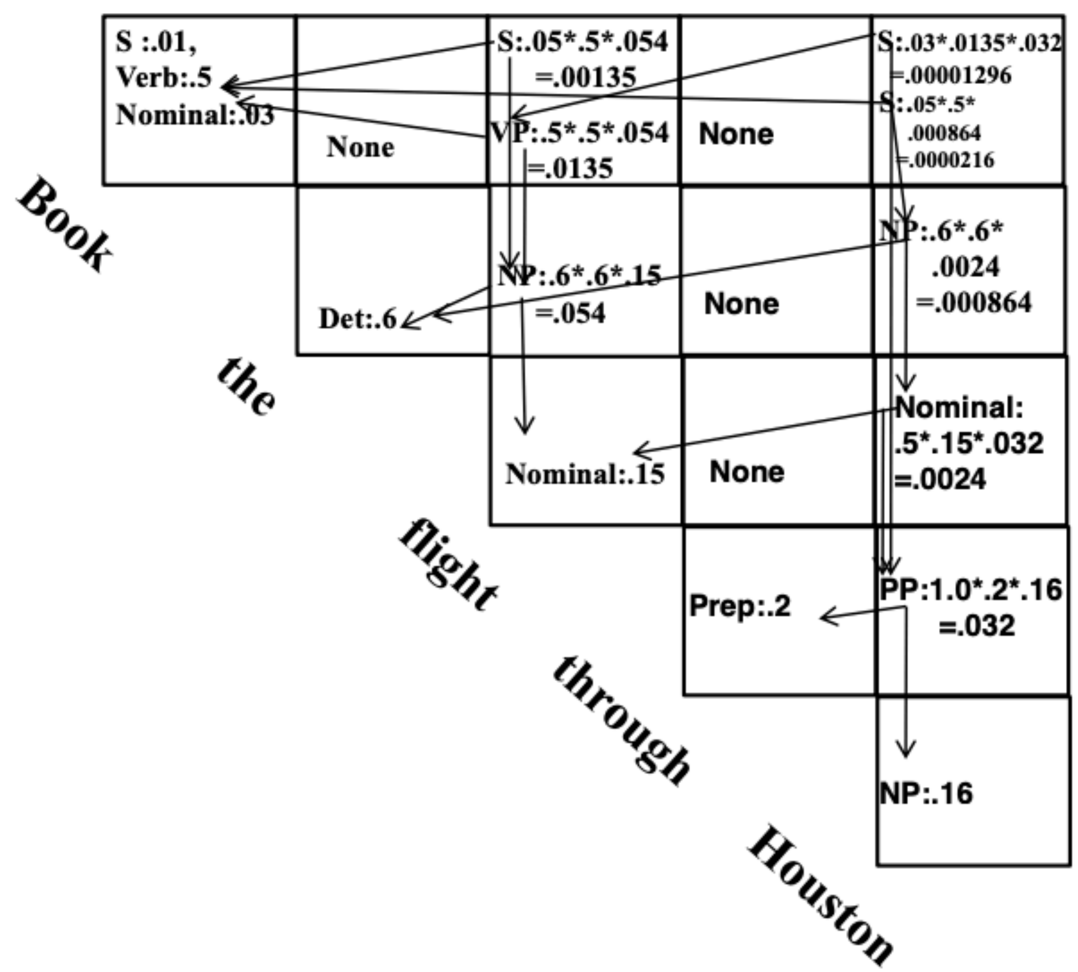


The CKY algorithm

- For all (i, j) such that $1 \leq i < j \leq n$ for all $X \in N$,

$$\pi(i, j, X) = \max_{X \rightarrow YZ \in R, i \leq k < j} q(X \rightarrow YZ) \times \pi(i, k, Y) \times \pi(k+1, j, Z)$$

Also stores backpointers which allow us to recover the parse tree



The CKY algorithm

Input: a sentence $s = x_1 \ldots x_n$, a PCFG $G = (N, \Sigma, S, R, q)$.
Initialization:
For all $i \in \{1 \ldots n\}$, for all $X \in N$,

$$\pi(i, i, X) = \begin{cases} q(X \rightarrow x_i) & \text{if } X \rightarrow x_i \in R \\ 0 & \text{otherwise} \end{cases}$$

Algorithm:

- For $l = 1 \ldots (n-1)$
  - For $i = 1 \ldots (n-l)$
    - Set $j = i + l$
    - For all $X \in N$, calculate

$$\pi(i, j, X) = \max_{\substack{X \rightarrow Y Z \in R, \\ s \in \{i \ldots (j-1)\}}} \left(q(X \rightarrow Y Z) \times \pi(i, s, Y) \times \pi(s+1, j, Z) \right)$$

and

$$bp(i, j, X) = \arg \max_{\substack{X \rightarrow Y Z \in R, \\ s \in \{i \ldots (j-1)\}}} \left(q(X \rightarrow Y Z) \times \pi(i, s, Y) \times \pi(s+1, j, Z) \right)$$

Output: Return $\pi(1, n, S) = \max_{t \in \mathcal{T}(s)} p(t)$, and backpointers bp which allow recovery of $\arg\max_{t \in \mathcal{T}(s)} p(t)$.

Running time?

$O(n^3|R|)$

CKY with unary rules

- In practice, we also allow unary rules:

$$X \to Y \text{ where } X, Y \in N$$

conversion to/from the normal form is easier

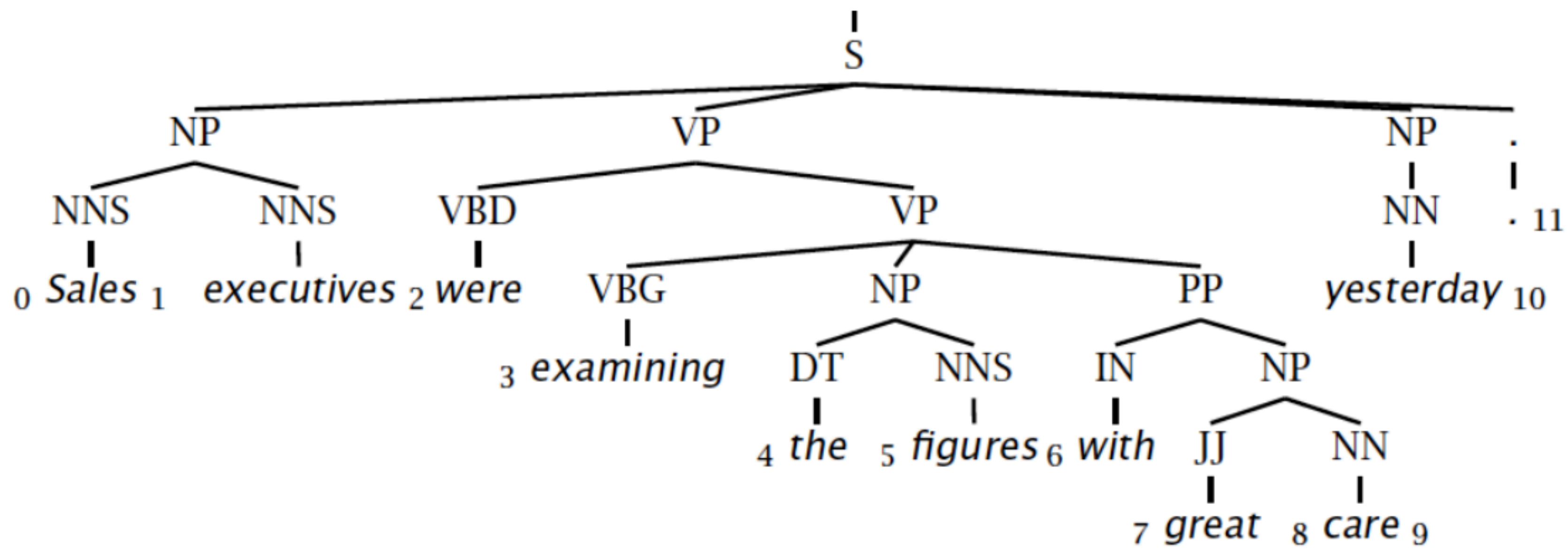
How does this change CKY?

$$\pi(i, j, X) = \max_{X \to Y \in R} q(X \to Y) \times \pi(i, j, Y)$$

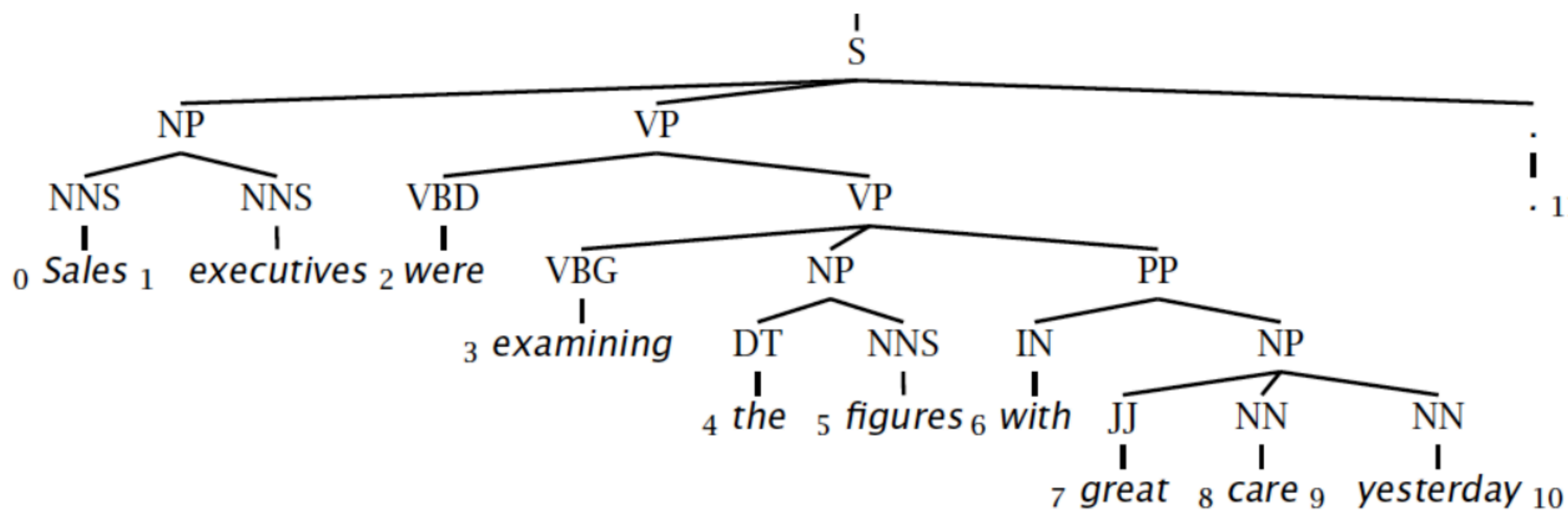
- Compute unary closure: if there is a rule chain $X \to Y_1, Y_1 \to Y_2, \ldots, Y_k \to Y$, add $q(X \to Y) = q(X \to Y_1) \times \cdots \times q(Y_k \to Y)$
- Update unary rule once after the binary rules

Evaluating constituency parsing

Gold standard brackets: **S-(0:11)**, **NP-(0:2)**, VP-(2:9), VP-(3:9), **NP-(4:6)**, PP-(6-9), NP-(7,9), NP-(9:10)



Candidate brackets: **S-(0:11)**, **NP-(0:2)**, VP-(2:10), VP-(3:10), **NP-(4:6)**, PP-(6-10), NP-(7,10)

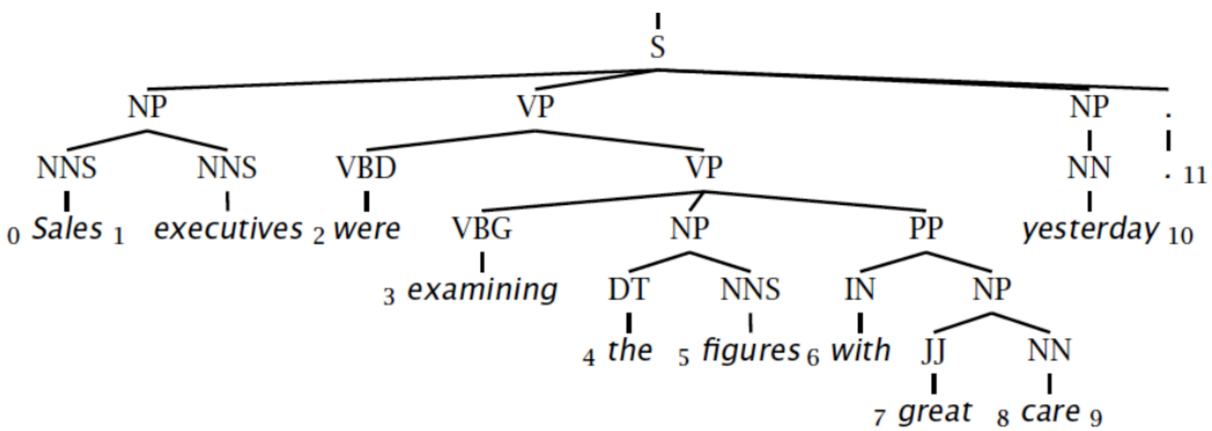


Evaluating constituency parsing

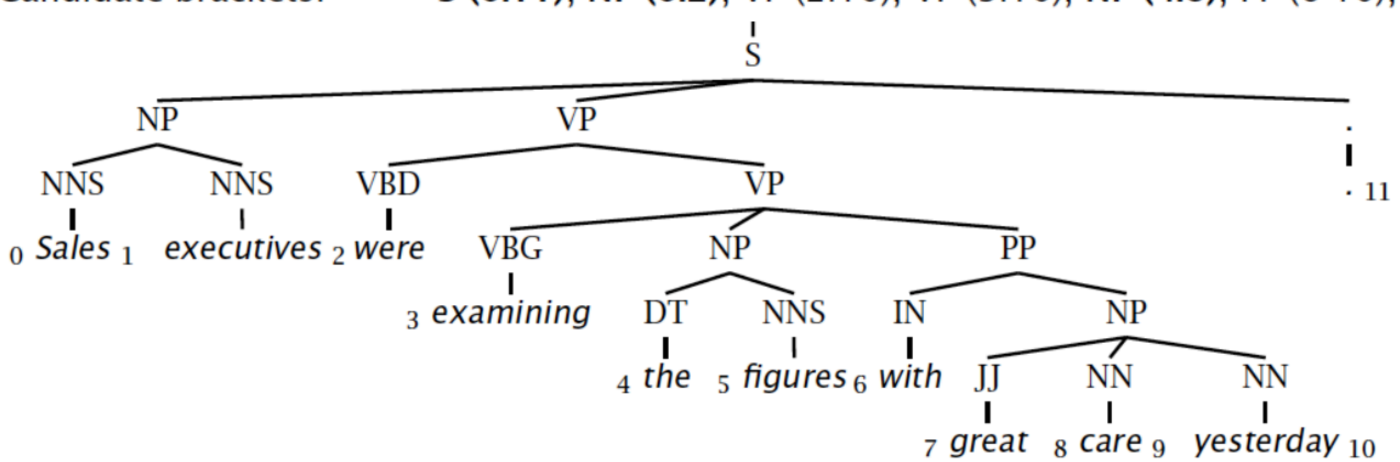
- Recall: (# correct constituents in candidate) / (# constituents in gold tree)
- Precision: (# correct constituents in candidate) / (# constituents in candidate)
- Labeled precision/recall require getting the non-terminal label correct
- F1 = (2 * precision * recall) / (precision + recall)

Evaluating constituency parsing

Gold standard brackets: **S-(0:11)**, **NP-(0:2)**, VP-(2:9), VP-(3:9), **NP-(4:6)**, PP-(6-9), NP-(7,9), NP-(9:10)



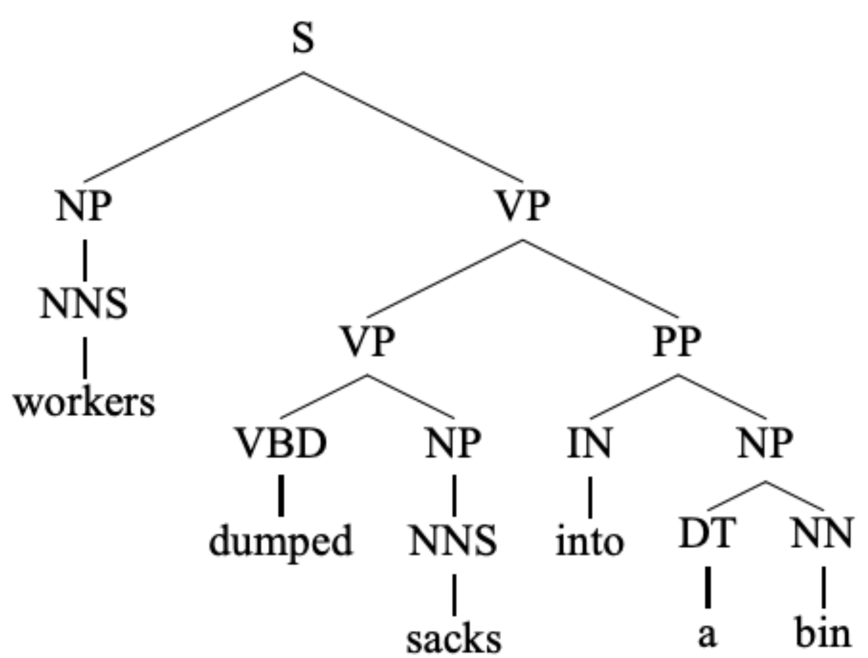
Candidate brackets: **S-(0:11)**, **NP-(0:2)**, VP-(2:10), VP-(3:10), **NP-(4:6)**, PP-(6-10), NP-(7,10)

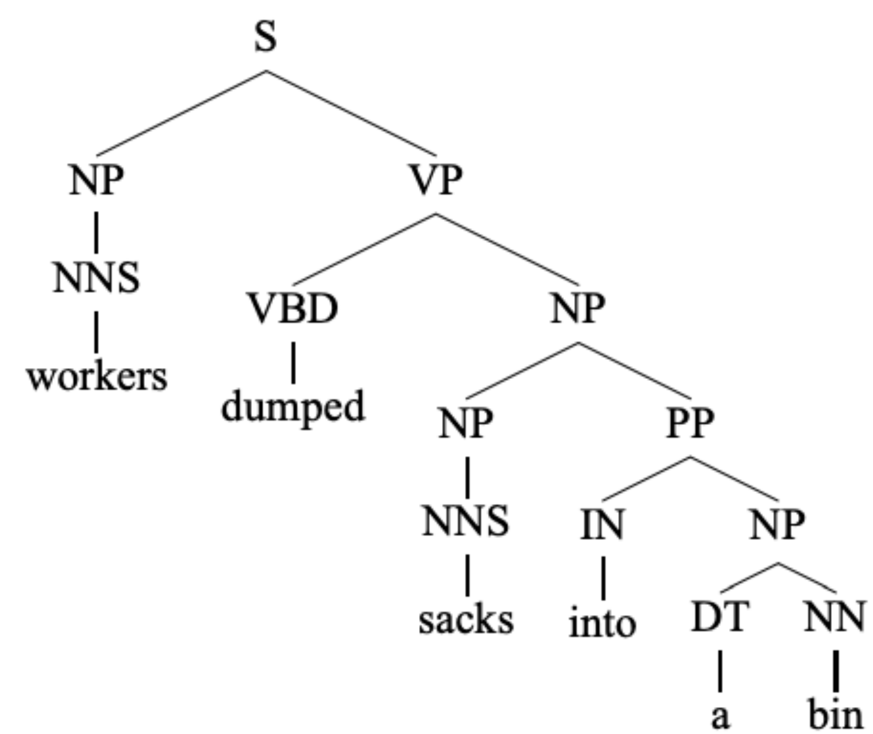


- Precision: 3/7 = 42.9%
- Recall: 3/8 = 37.5%
- F1 = 40.0%
- Tagging accuracy: 100%

Weaknesses of PCFGs

- Lack of sensitivity to lexical information (words)





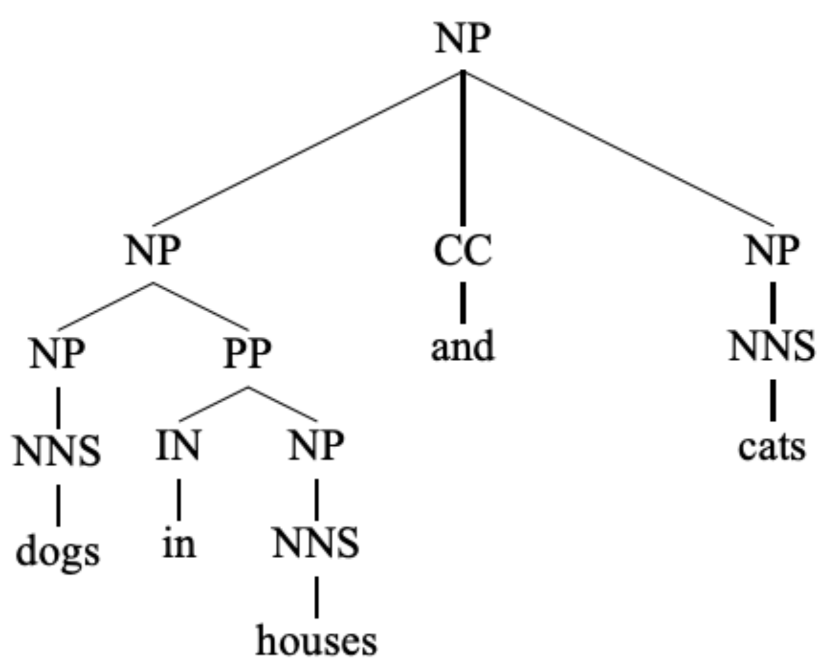
The only difference between these two parses:

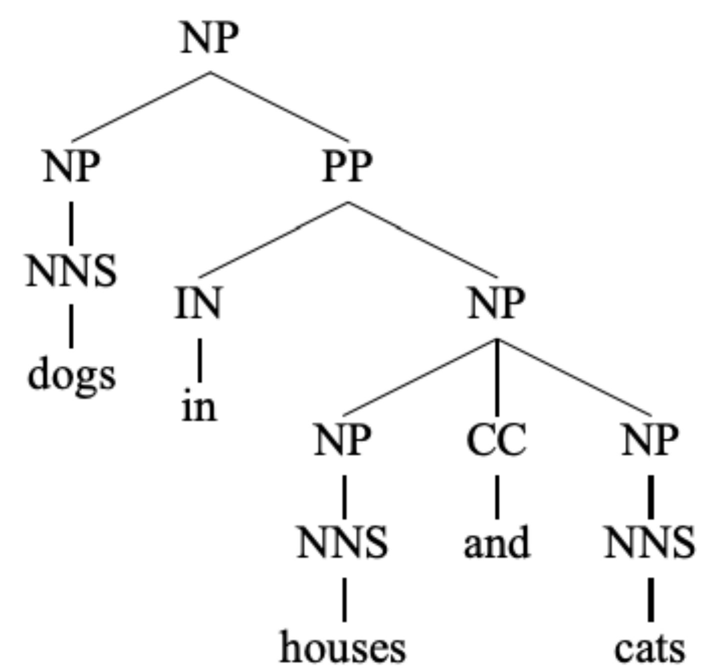
$q(\text{VP} \rightarrow \text{VP PP})$ vs $q(\text{NP} \rightarrow \text{NP PP})$

... without looking at the words!

Weaknesses of PCFGs

- Lack of sensitivity to lexical information (words)

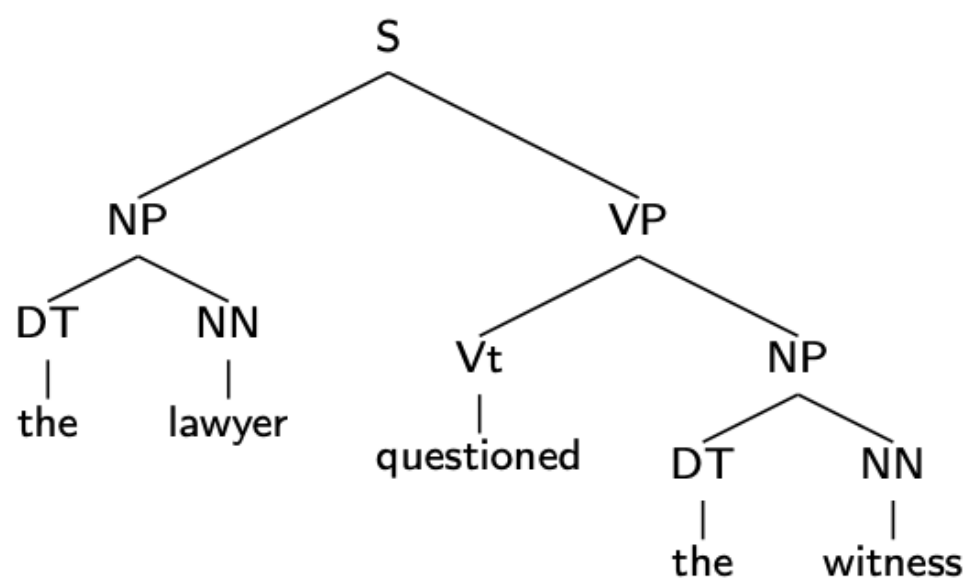


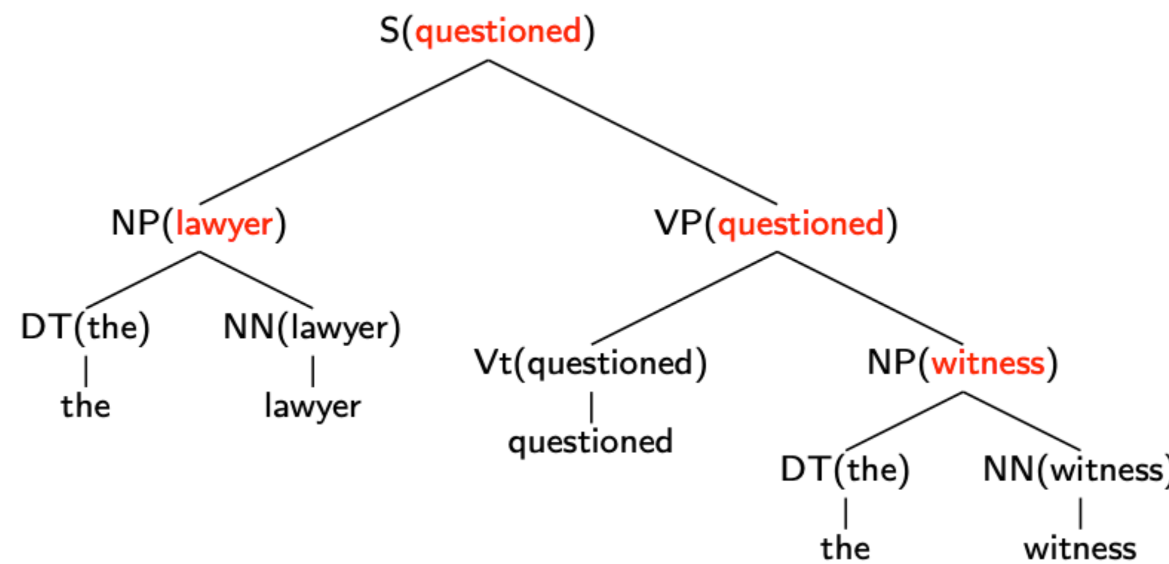


Exactly the same set of context-free rules!

Lexicalized PCFGs

- Key idea: add **headwords** to trees





- Each context-free rule has one special child that is the head of the rule (a core idea in syntax)

S	$\Rightarrow$	NP	VP		(VP is the head)
VP	$\Rightarrow$	Vt	NP		(Vt is the head)
NP	$\Rightarrow$	DT	NN	NN	(NN is the head)

Lexicalized PCFGs

S(saw)	$\rightarrow_2$	NP(man)	VP(saw)
VP(saw)	$\rightarrow_1$	Vt(saw)	NP(dog)
NP(man)	$\rightarrow_2$	DT(the)	NN(man)
NP(dog)	$\rightarrow_2$	DT(the)	NN(dog)
Vt(saw)	$\rightarrow$	saw	
DT(the)	$\rightarrow$	the	
NN(man)	$\rightarrow$	man	
NN(dog)	$\rightarrow$	dog	

- Further reading: *Michael Collins. 2003. Head-Driven Statistical Models for Natural Language Parsing*.
- Results for a PCFG: 70.6% recall, 74.8% precision
- Results for a lexicalized PCFG: 88.1% recall, 88.3% precision

http://nlpprogress.com/english/constituency_parsing.html