Lecture 10: Grammar-based Parsing

Constituency Parsing and Context-Free Grammars as a Test Case

Why is Parsing Hard?

- An exponential number of options per sentence (+labels differ between examples)
- Ambiguity
- Many many rules to learn
  - Some common ones, like those we discussed, and some very specific (power-law again)
  - Examples of less common rules:
    - River names (“River Thames”, “Mississippi River”)
    - Absolute construction (“Barring bad weather, we’re going to the beach tomorrow”)
- Language speakers have no **explicit** knowledge of the rules of grammar

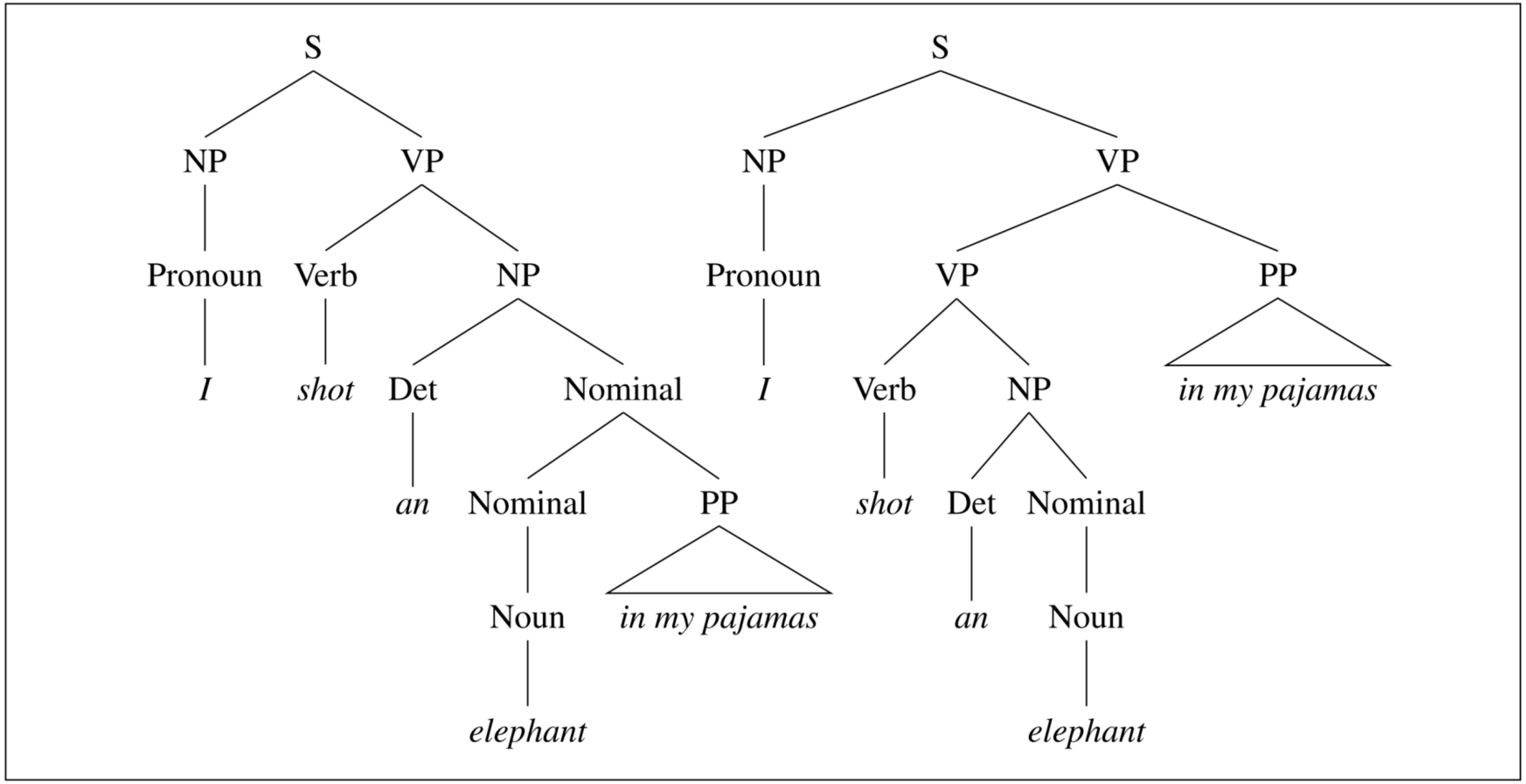
Pre 1990 (“Classical”) NLP Parsing

- Wrote symbolic grammar (CFG or often richer) and lexicon
- Used grammar/proof systems to find derivations (parses) for sentences
- This scaled very badly and had poor coverage (grammars tend to accept many incorrect parses)
- Example: *“Fed raises interest rates 0.5% in effort to control inflation”*
  - Minimal grammar: 36 parses
  - Simple 10 rule grammar: 592 parses
  - Real-size broad-coverage grammar: millions of parses

Miniature Grammar for English

Grammar	**Lexicon**
$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid the \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Pronoun$	$Pronoun \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$Proper\text{-}Noun \rightarrow Houston \mid NWA$
$NP \rightarrow Det\ Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	$Preposition \rightarrow from \mid to \mid on \mid near \mid through$
$Nominal \rightarrow Nominal\ Noun$	…
$Nominal \rightarrow Nominal\ PP$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	
$VP \rightarrow Verb\ NP\ PP$	
$VP \rightarrow Verb\ PP$	
$VP \rightarrow VP\ PP$	
$PP \rightarrow Preposition\ NP$	

PP Attachment Ambiguity



But also the unambiguous “I shot an elephant over lunch” would get two parses

Deterministic Grammars and Their Limitations

- Categorical constraints can be added to grammars to limit unlikely/weird parses for sentences
- But the attempt makes the grammars not robust
  - In traditional systems, commonly 30% of sentences would have no parse (worse in unedited text)
- A less constrained grammar can parse more sentences
  - But simple sentences end up with ever more parses with no way to choose between them
- We need mechanisms that allow us to find the most likely parse(s) for a sentence
  - Statistical parsing lets us work with very loose grammars that admit millions of parses for sentences but still quickly find the best parse(s)

The Rise of Annotated Data: The Penn Treebank (early 90s)

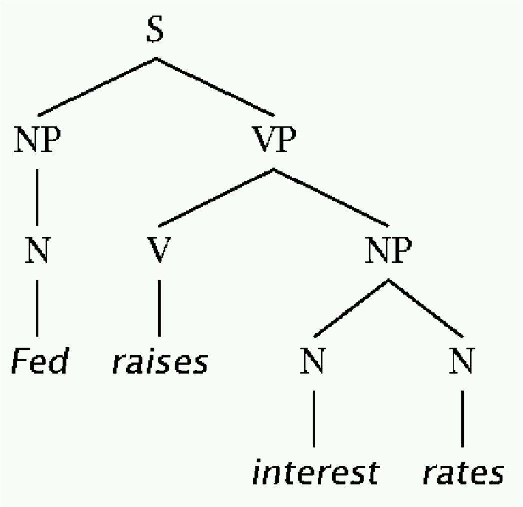
```
( (S
    (NP-SBJ (DT The) (NN move))
    (VP (VBD followed)
      (NP
        (NP (DT a) (NN round))
        (PP (IN of)
          (NP
            (NP (JJ similar) (NNS increases))
            (PP (IN by)
              (NP (JJ other) (NNS lenders)))
            (PP (IN against)
              (NP (NNP Arizona) (JJ real) (NN estate) (NNS loans))))))
      (, ,)
      (S-ADV
        (NP-SBJ (-NONE- *))
        (VP (VBG reflecting)
          (NP
            (NP (DT a) (VBG continuing) (NN decline))
            (PP-LOC (IN in)
              (NP (DT that) (NN market)))))))
    (. .)))
```

The Rise of Annotated Data

- Starting off, building a treebank seemed a lot slower and less useful than building a grammar
- But a treebank gives us many things
  - Reusability of the labor: many parsers, POS taggers, etc., valuable resource for linguistics
  - Broad coverage
  - Frequencies and distributional information
  - A way to evaluate systems
- Penn Treebank WSJ: 50K sentences, 40K training, 2400 test

Constituency Parsing and Context-free Grammars (CFGs)

- Writing parsing rules:
  - N → Fed
  - V → raises
  - NP → N
  - S → NP VP
  - VP → V NP
  - NP → N N
  - NP → NP PP
  - N → interest
  - N → raises



Writing the Rules of Grammar: Context-free Grammars

- A context-free grammar is a tuple <*N*, Σ , *S*, *R*>
  - *N* : the set of non-terminals
    - Phrasal categories: S, NP, VP, ADJP, etc.
    - Parts-of-speech (pre-terminals): NN, JJ, DT, VB
  - Σ : the set of terminals (the words)
  - *S* : the start symbol
    - Often written as ROOT or TOP
    - ***Not*** usually the sentence non-terminal S – why not?
  - *R* : the set of rules
    - Of the form $X \rightarrow Y_1\ Y_2 \ldots Y_n$, with $X \in \boldsymbol{N}$, $\boldsymbol{n \geq 0}$, $\boldsymbol{Y_i} \in (\boldsymbol{N} \cup \Sigma)$
    - Examples: S → NP VP, VP → VP CC VP
    - Also called rewrites, productions, or local trees

Example Grammar

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

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

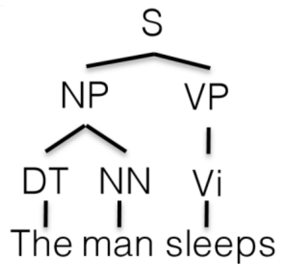
S=sentence, VP-verb phrase, NP=noun phrase, PP=prepositional phrase, DT=determiner, Vi=intransitive verb, Vt=transitive verb, NN=noun, IN=preposition

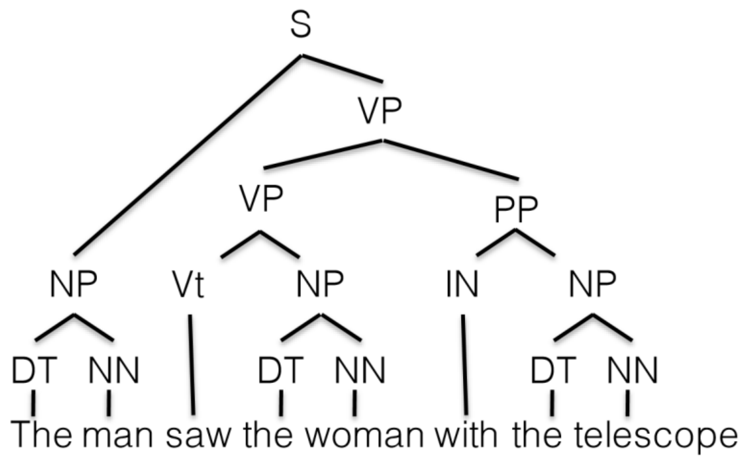
Example Parse

$R =$

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
DT	$\Rightarrow$	the
IN	$\Rightarrow$	with
IN	$\Rightarrow$	in





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

A Parse Tree as a CFG Derivation

- Deterministic Parsing:
  - Given a grammar (say a CFG grammar)
  - Given a sentence
  - Find a derivation that yields the sentence in its leaves

- This can be done with dynamic programming (you may remember it from a course on formal languages); we will see a more general solution to this problem next lesson

Probabilistic Context-free Grammars (PCFG)

- A context-free grammar is a tuple <*N*, Σ , *S*, *R*>
  - *N* : the set of non-terminals
    - Phrasal categories: S, NP, VP, ADJP, etc.
    - Parts-of-speech (pre-terminals): NN, JJ, DT, VB
  - Σ : the set of terminals (the words)
  - *S* : the start symbol
    - Often written as ROOT or TOP
    - *Not* usually the sentence non-terminal S
  - *R* : the set of rules
    - Of the form $X \rightarrow Y_1\ Y_2 \ldots Y_n$, with $X \in N$, $n \geq 0$, $Y_i \in (N \cup \Sigma)$
    - Examples: $S \rightarrow NP\ VP$, $VP \rightarrow VP\ CC\ VP$
    - Also called rewrites, productions, or local trees
- A PCFG adds a distribution q:
  - Probability q(r) for each $r \in R$, *such that* for all $X \in N$:

$$\sum_{\alpha\rightarrow\beta\in R:\alpha=X} q(\alpha \rightarrow \beta) = 1$$

PCFG Example

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

- Probability of a tree t with rules

$$\alpha_1 \rightarrow \beta_1, \alpha_2 \rightarrow \beta_2, \ldots, \alpha_n \rightarrow \beta_n$$

is

$$p(t) = \prod_{i=1}^{n} q(\alpha_i \rightarrow \beta_i)$$

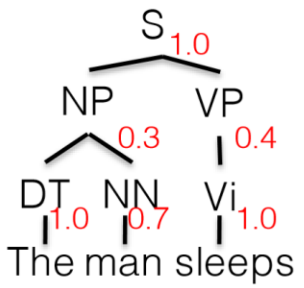
where $q(\alpha \rightarrow \beta)$ is the probability for rule $\alpha \rightarrow \beta$.

PCFG Example

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

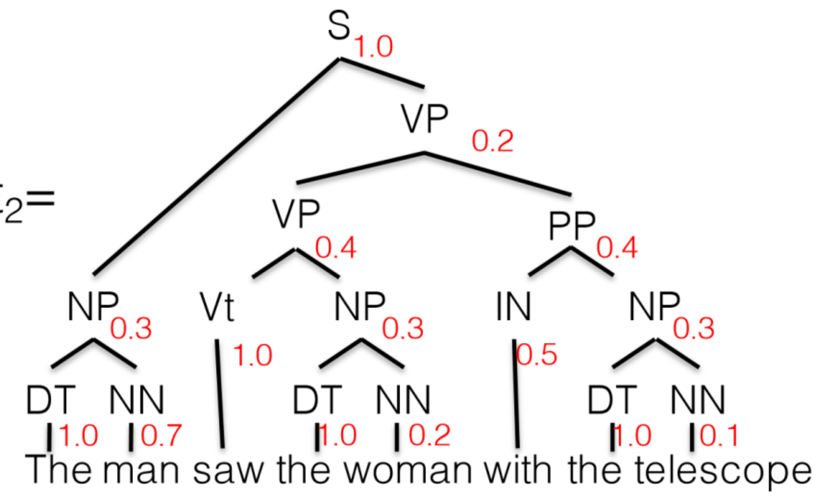
Vi	$\Rightarrow$	sleeps	1.0
Vt	$\Rightarrow$	saw	1.0
NN	$\Rightarrow$	man	0.7
NN	$\Rightarrow$	woman	0.2
NN	$\Rightarrow$	telescope	0.1
DT	$\Rightarrow$	the	1.0
IN	$\Rightarrow$	with	0.5
IN	$\Rightarrow$	in	0.5

$t_1=$



$p(t_1)=1.0*0.3*1.0*0.7*0.4*1.0$

$t_2=$



$p(t_s)=1.0*0.3*1.0*0.7*0.2*0.4*1.0*0.3*1.0*0.2*0.4*0.5*0.3*1.0*0.1$

Grammar-based vs. Discriminative Approaches to Parsing

- Generative models:
  - This is important, e.g., for unsupervised learning
  - Easier to train
  - In some cases, we want to assume (e.g., for cognitive plausibility) that the distribution of the observed variables is modeled as well
- Models tend to be more interpretable: consists of probabilistic rules
- Models are often easier to reason over, because they have more structure
  - For instance, there are many theoretical results on what type of phenomena CFG and PCFG may or may capture

Learning and Inference

- Model:
  - The probability of a tree t with n rules $\alpha_i \rightarrow \beta_i$, $i=1..n$

$$p(t) = \prod_{i=1}^{n} q(\alpha_i \rightarrow \beta_i)$$

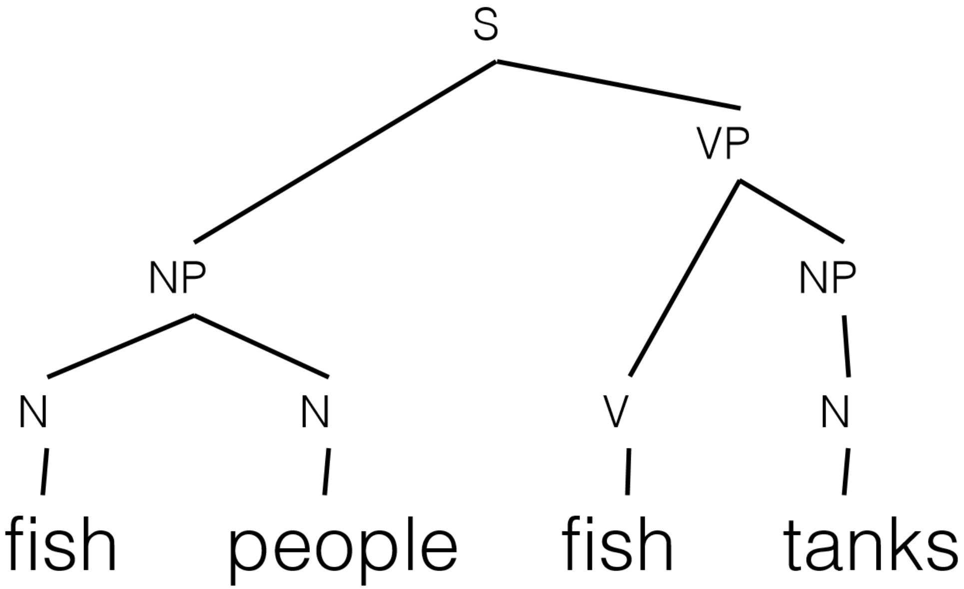
- Learning
  - Read the rules off of labeled sentences, use ML estimates for probabilities
  - and use all of our standard smoothing tricks!

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

- Inference
  - For input sentence s, define $T(s)$ to be the set of trees whose yield is s (whose leaves, read left to right, match the words in s)

$$t^*(s) = \arg\max_{t \in \mathcal{T}(s)} p(t)$$

The Constituency Parsing Problem



PCFG

Rule	Prob $\boldsymbol{\theta_i}$
$S \rightarrow NP\ VP$	θ_0
$NP \rightarrow NP\ NP$	θ_1
…	
$N \rightarrow$ fish	θ_{42}
$N \rightarrow$ people	θ_{43}
$V \rightarrow$ fish	θ_{44}
…	

Chomsky Normal Form (CNF)

- It's often useful to treat all CFG rules as binary
- A common binarization is the Chomsky normal form, which is a grammar where each production is either of the form $A \rightarrow B\ C$ or $A \rightarrow w$, where *A, B and C* are pre-terminals and *w* is a terminal
- Every CFG grammar can be converted to a CNF, which includes the same sentences ("weak equivalence")
- Example:

$A \rightarrow B\ C\ D\ E$

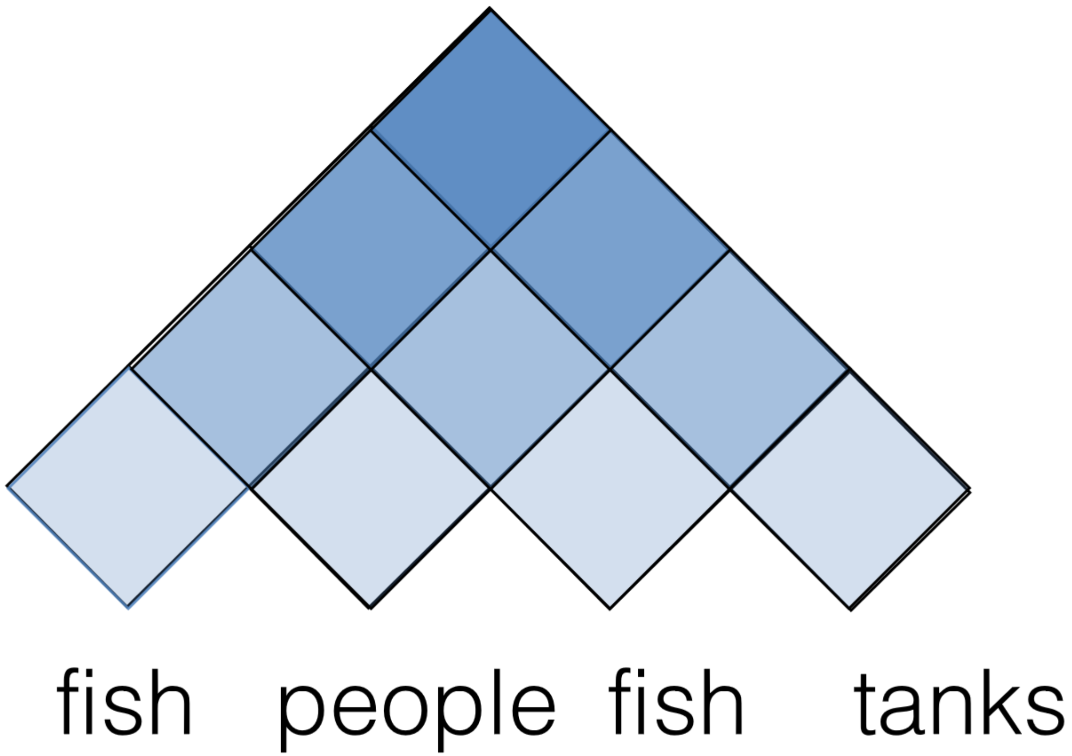
$A \rightarrow A_1\ E$
$A_1 \rightarrow A_2\ D$
$A_2 \rightarrow B\ \ C$

A Recursive Parser

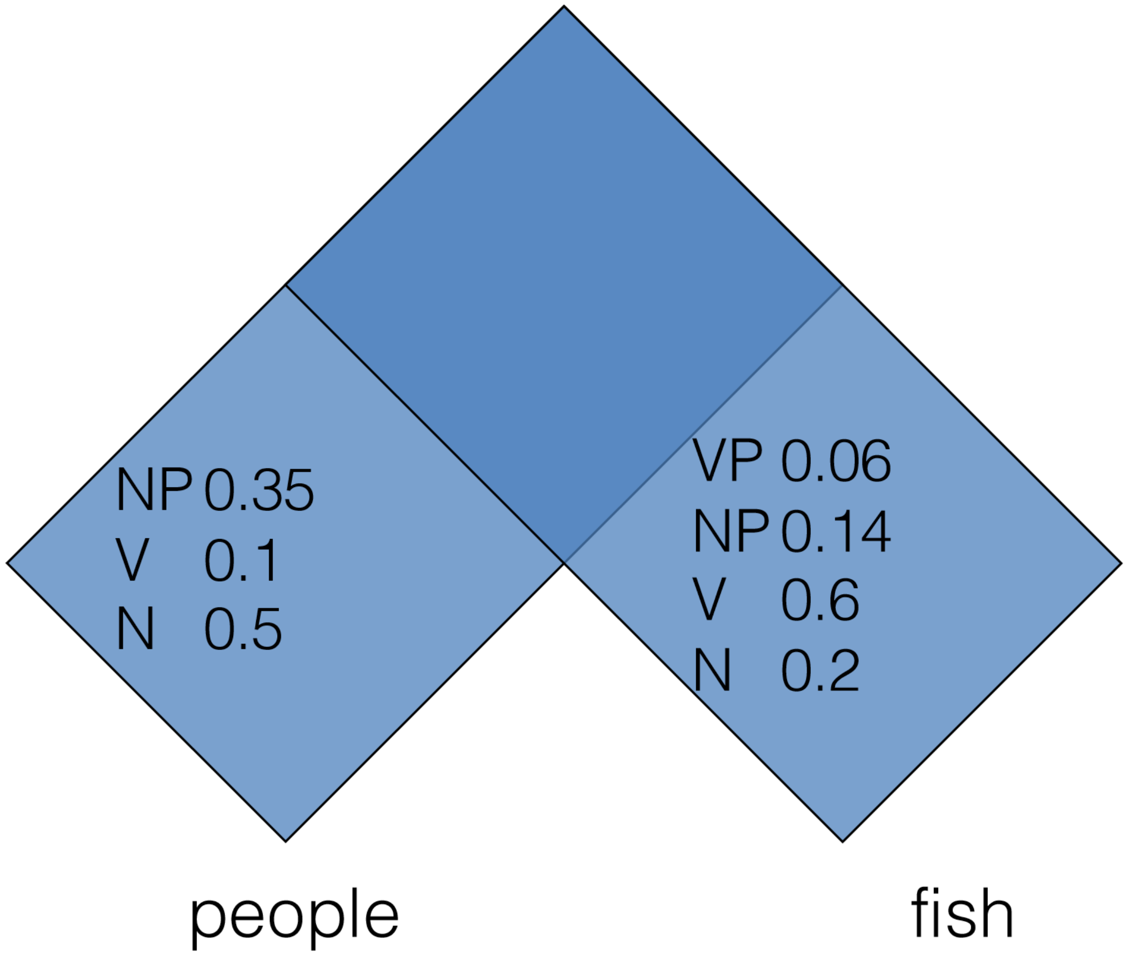
```
bestScore(X,i,j,s)
   if (j == i)
        return q(X->s[i])
   else
        return max q(X->YZ) *
                              bestScore(Y,i,k,s) *
                              bestScore(Z,k+1,j,s)
```

- Will this parser work?
- Why or why not?
- Q: Remind you of anything? Can we adapt this to other models / inference tasks?

Cocke-Kasami-Younger (CKY) Constituency Parsing

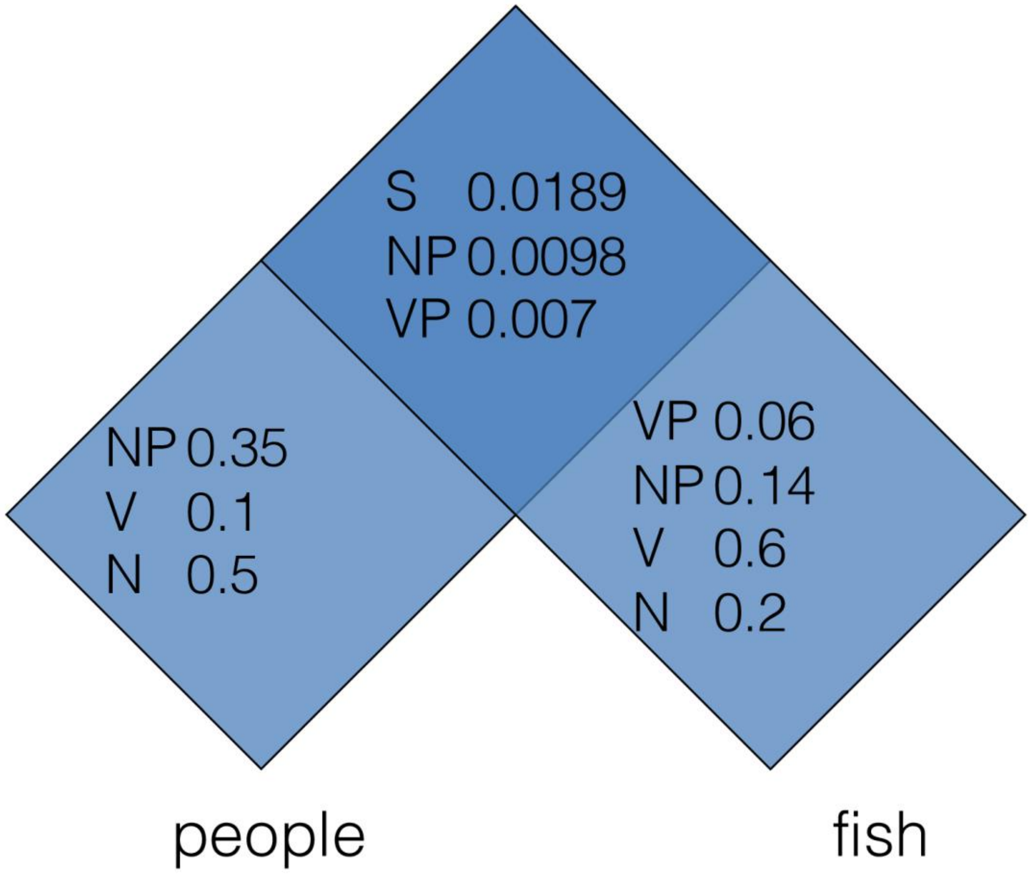


Cocke-Kasami-Younger (CKY) Constituency Parsing



S → NP VP	0.9
VP → V NP	0.5
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
PP → P NP	1.0

Cocke-Kasami-Younger (CKY) Constituency Parsing



S → NP VP	0.9
VP → V NP	**0.5**
VP → V @VP_V	0.3
VP → V PP	0.1
@VP_V → NP PP	1.0
NP → NP NP	0.1
NP → NP PP	0.2
PP → P NP	1.0

The CKY Algorithm

- Input: a sentence s = x_1 .. x_n and a PCFG = <*N*, Σ ,*S*, *R*, *q*>
- Initialization: For i = 1 … n and all X in N

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

- For l = 1 … (n-1) [iterate all phrase lengths]
  - For i = 1 … (n-l) and j = i+l [iterate all phrases of length l]
    - For all X in N [iterate all non-terminals]

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

    - also, store back pointers

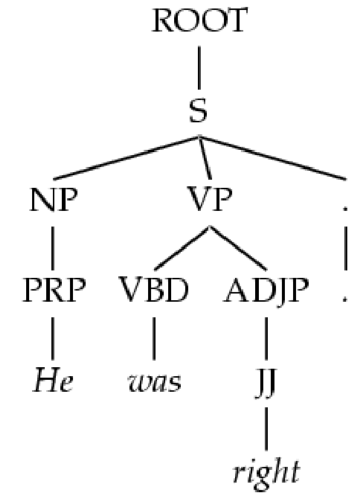
$$bp(i,j,X) = \arg \max_{\substack{X \rightarrow YZ \in R, \\ s \in \{i \ldots (j-1)\}}} \left(q(X \rightarrow YZ) \times \pi(i,s,Y) \times \pi(s+1,j,Z) \right)$$

Learning: Estimating a PCFG from a Treebank

- The ML estimator can just be computed by keeping track of the frequency distribution of derivations:

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

- Example:



ROOT → S 1

S → NP VP . 1

NP → PRP 1

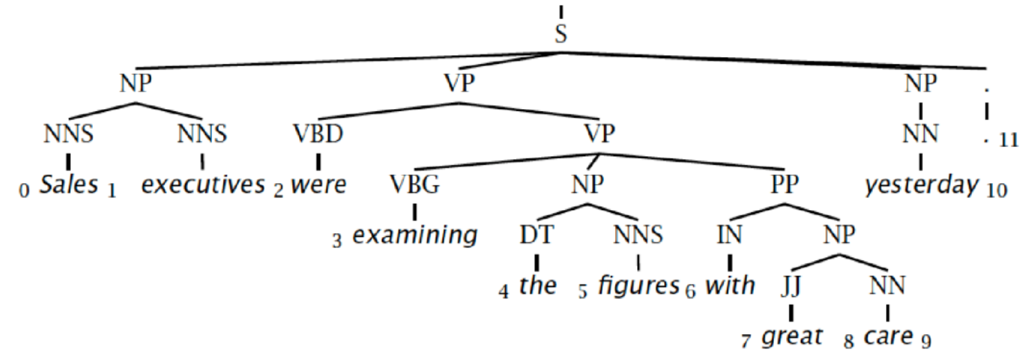
VP → VBD ADJP 1

…..

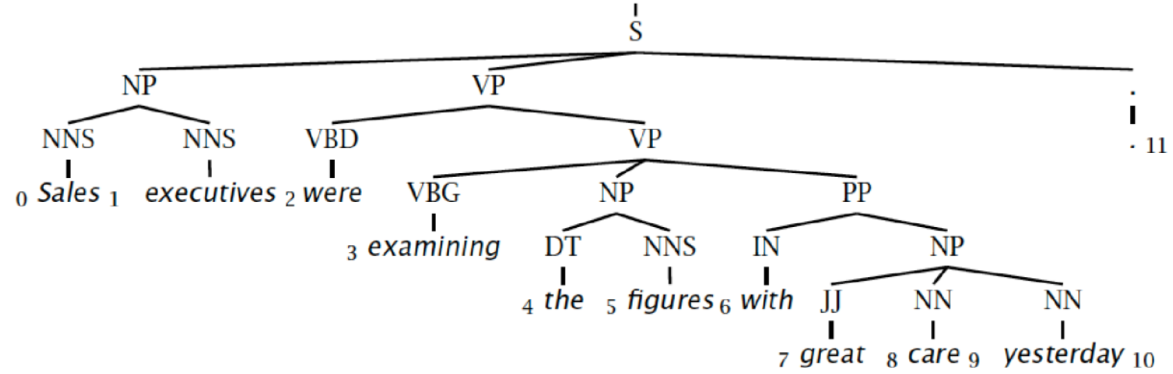
Evaluation

- Comparing the predicted trees to the gold standard:

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)



Evaluation

- Constituents match if they cover the same span of tokens and have the same label
- Compute Recall/Precision/F1:

$$P = \frac{\#\text{Matching Constituents}}{\#\text{Predicted Constituents}}$$

$$R = \frac{\#\text{Matching Constituents}}{\#\text{Gold Constituents}}$$

$$F_1 = \frac{2 \cdot P \cdot R}{P + R}$$

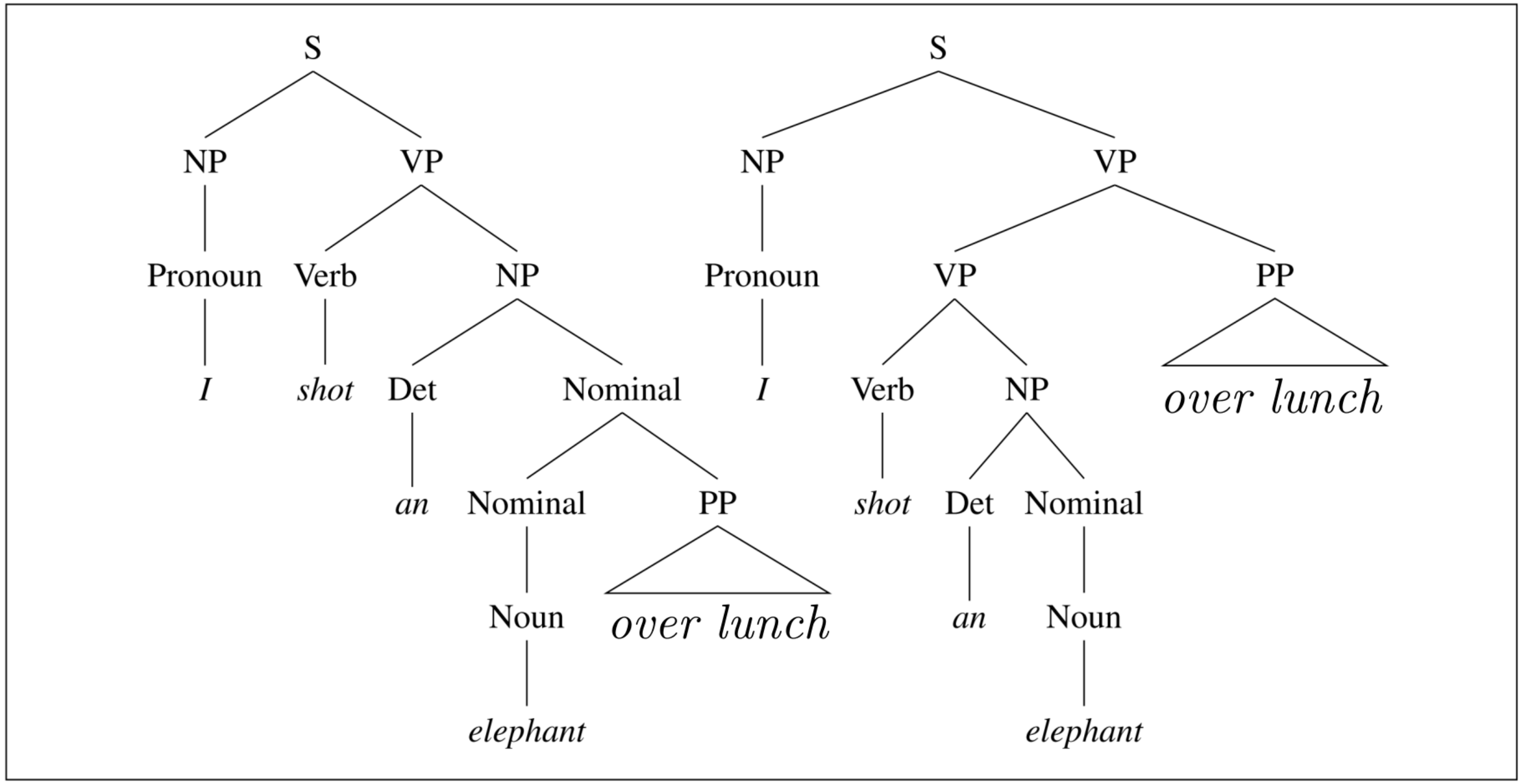
- Unlabeled Precision/Recall/F1: the same without requiring that the labels be the same for constituents to match

How good are PCFGs?

- “Vanilla” PCFGs score about 73% F1 on the Penn Treebank
  - This is quite low
- The problem is that the independence assumptions are very strong
  - Each production is independent of any other production
- Examples of problems:
  - All verb phrases (VPs) have the same distribution of subjects
  - All Noun Phrases have the same distribution (regardless of their role in the sentence)
  - All clauses have the same distribution (regardless of the other clauses in the sentence)

Recall: PP Attachment Ambiguity

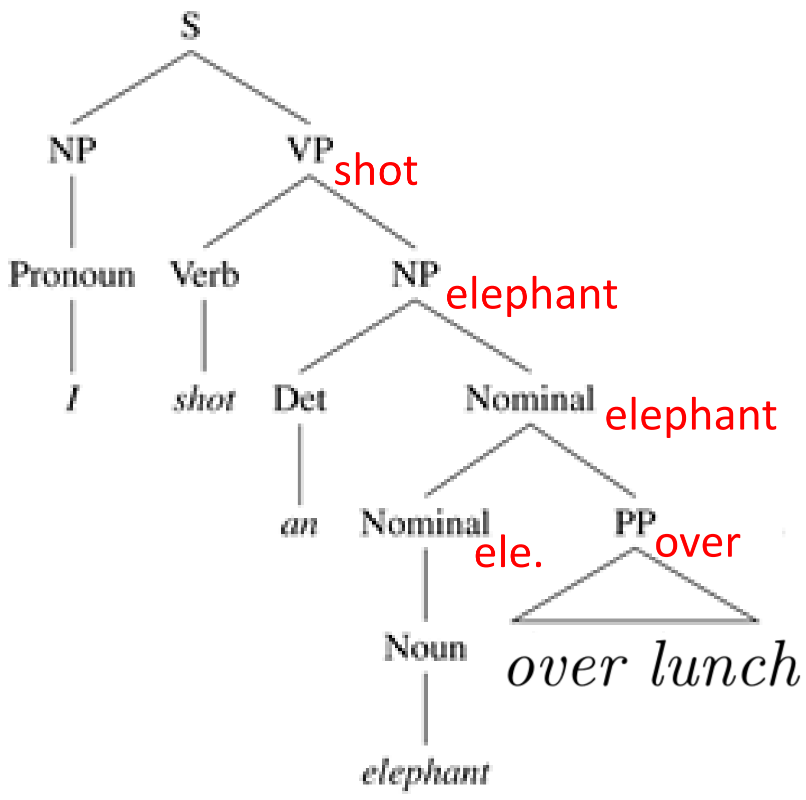
- Which tree receives a higher probability is only determined by the relative likelihood of
  - Nominal → Nominal PP
  - VP → VP PP
- However, “elephant” usually doesn’t take an “over” PP, while “shot” might take an “over” PP as in this example



Head Lexicalization

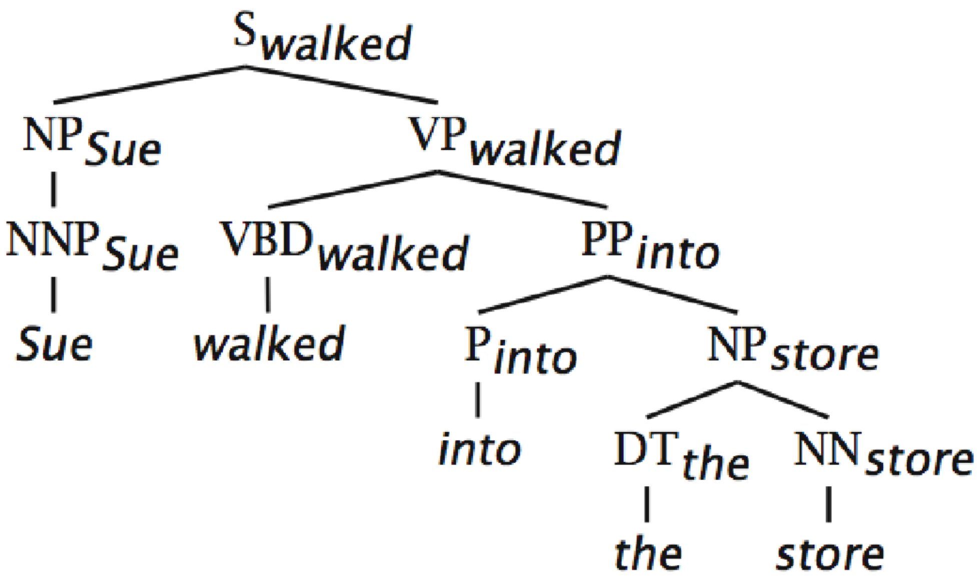
- Add "headwords" to each phrasal node
- In the example the two rules are now:
  - $\text{Nominal}_{\text{elephant}} \rightarrow \text{Nominal}_{\text{elephant}}\ \text{PP}_{\text{over}}$
  - $\text{VP}_{\text{shot}} \rightarrow \text{VP}_{\text{shot}}\ \text{Pp}_{\text{over}}$

- The first rule is likely to get a low probability
- Headwords are not annotated in most treebanks, so we usually use heuristics for finding heads, e.g.:
  - NP: Take leftmost NP, take rightmost N, take rightmost JJ, take right child
  - VP: Take leftmost V, take leftmost VP, take left child



Head Lexicalization of PCFGs

- The head word of a phrase gives a good representation of the phrase's structure and meaning
- Puts the properties of words back into a PCFG
  - $VP_{walked} \rightarrow VBD_{walked}\ PP_{into}$ could have a high score
  - $VP_{walked} \rightarrow VBD_{walked}\ PP_{about}$ could have a lower score



Price of Lexicalization

- Lexicalization can resolve ambiguities and lead to huge boost in performance
- But, what’s the cost?
- More rules lead to:
  - Higher computational demands (CKY: $O(|Rules|n^3)$)
  - More difficult to get good estimates of probabilities

    $VP_{walked} \rightarrow VBD_{walked} \;\; PP_{into}$

  - Smoothing techniques, similar to what we saw for language models alleviate this difficulty

The Collins Parser (1997)

- Very influential lexicalized PCFG models that alleviate sparsity (we show Model 1)
- The probability of the rule $rule_l \rightarrow rule_r$ of the form

$$P(\mathbf{h}) \rightarrow L_n(l_n), \ldots, L_1(l_1), \mathbf{H}(\mathbf{h}), R_1(r_1), \ldots, R_m(r_m)$$

$$\mathcal{P}(rule_r | rule_l) = \mathcal{P}(L_n(l_n), \ldots, L_1(l_1), \mathbf{H}(\mathbf{h}), R_1(r_1), \ldots, R_m(r_m) | P(\mathbf{h}))$$

$$= \mathcal{P}(\mathbf{H}(\mathbf{h}) \mid P(\mathbf{h})) \cdot \prod_{i=1}^{n+1} \mathcal{P}(L_i(l_i) \mid P(\mathbf{h}), \mathbf{H}(\mathbf{h})) \cdot \prod_{i=1}^{m+1} \mathcal{P}(R_i(r_i) \mid P(\mathbf{h}), \mathbf{H}(\mathbf{h}))$$

P: the parent constituent
h: the headword of *P*
$L_1,\ldots,L_n$: the constituents to the left of the head
$l_1,\ldots,l_n$: the headwords of $L_1,\ldots,L_n$
$R_1,\ldots,R_m$: the constituents to the right of the head
$r_1,\ldots,r_m$: the headwords of $L_1,\ldots,L_m$
H: the child of P that contains h

Michael Collins. 2003. Head-Driven Statistical Models for Natural Language Parsing. *In Computational Linguistics.*

The Collins Parser (1997)

$$\mathcal{P}(rule_r|rule_l) = \mathcal{P}(L_n(l_n), \ldots, L_1(l_1), \mathbf{H}(\mathbf{h}), R_1(r_1), \ldots, R_m(r_m) \,|\, P(\mathbf{h}))$$

$$= \mathcal{P}(\mathbf{H}(\mathbf{h}) \mid P(\mathbf{h})) \quad \cdot \quad \prod_{i=1}^{n+1} \mathcal{P}(L_i(l_i) \mid P(\mathbf{h}), \mathbf{H}(\mathbf{h}))$$

$$\cdot \quad \prod_{i=1}^{m+1} \mathcal{P}(R_i(r_i) \mid P(\mathbf{h}), \mathbf{H}(\mathbf{h}))$$

- In words: the probability of generating from a parent with category *P* and headword *h*, the constituents with categories $L_n,\ldots,L_1,H,R_1,\ldots,R_m$ and corresponding headwords $l_n,\ldots,l_1,h,r_1,\ldots,r_m$ is given by the above expression
- The generation ends when $L_{m+1}=STOP$ and $R_{n+1}=STOP$

The Collins Parser (1997)

$$\mathcal{P}(rule_r|rule_l) = \mathcal{P}(L_n(l_n), \ldots, L_1(l_1), \mathbf{H}(\mathbf{h}), R_1(r_1), \ldots, R_m(r_m) \,|\, P(\mathbf{h}))$$

$$= \mathcal{P}(\mathbf{H}(\mathbf{h}) \mid P(\mathbf{h})) \quad \cdot \quad \prod_{i=1}^{n+1} \mathcal{P}(L_i(l_i) \mid P(\mathbf{h}), \mathbf{H}(\mathbf{h}))$$

$$\cdot \quad \prod_{i=1}^{m+1} \mathcal{P}(R_i(r_i) \mid P(\mathbf{h}), \mathbf{H}(\mathbf{h}))$$

- This model still has many parameters, but it reduces each of the terms in the product to depend on 2-3 lexicalized categories, instead of n+m+2 for a regular lexicalized PCFG of this model

$$P(\mathbf{h}) \rightarrow L_n(l_n), \ldots, L_1(l_1), \mathbf{H}(\mathbf{h}), R_1(r_1), \ldots, R_m(r_m)$$

Learning and Inference

- **Learning:**
  - Maximum likelihood estimation is done by counting
  - As always, there are many many parameters, which creates sparsity issues
  - Back-off models are used to overcome sparsity
- **Inference:**
  - Inference is done using chart parsing, such as CKY
  - Due to the very large number of non-terminals, it is impossible to do exact inference
  - In practice, keeping only the highest scoring K options in each chart entry works well

Results

- Collins's first model improves F1 score on the Penn Treebank to over 87% (in-domain setting)
- Later improvements raised the score to about 93%