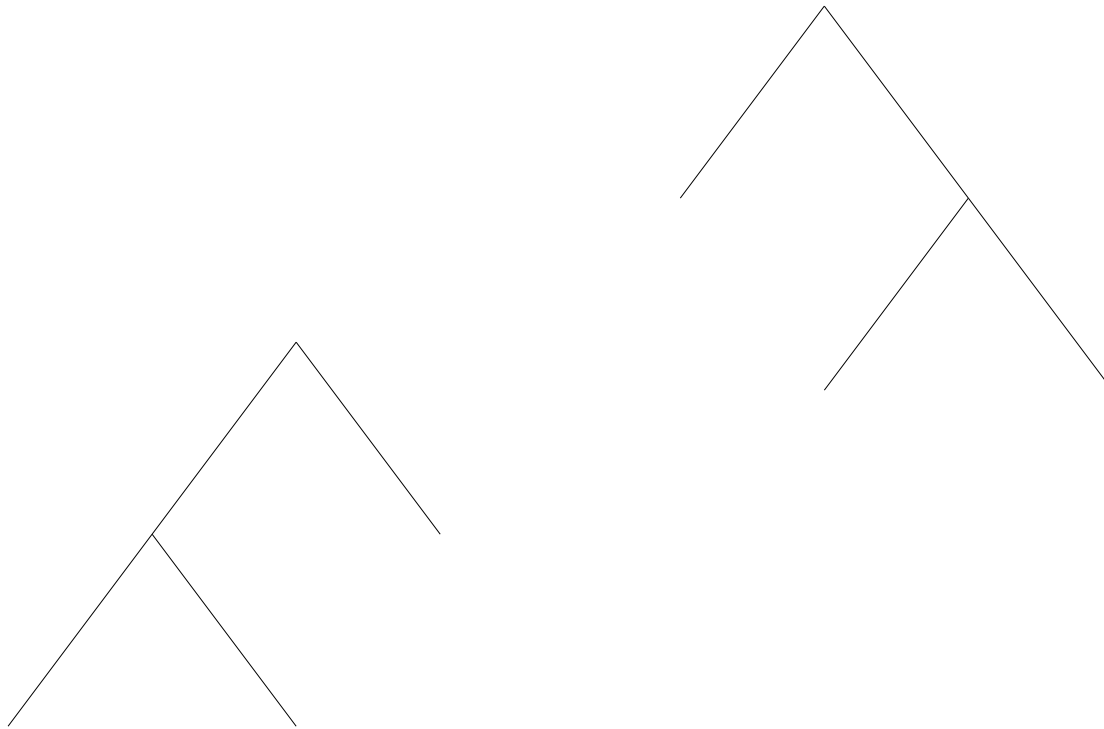


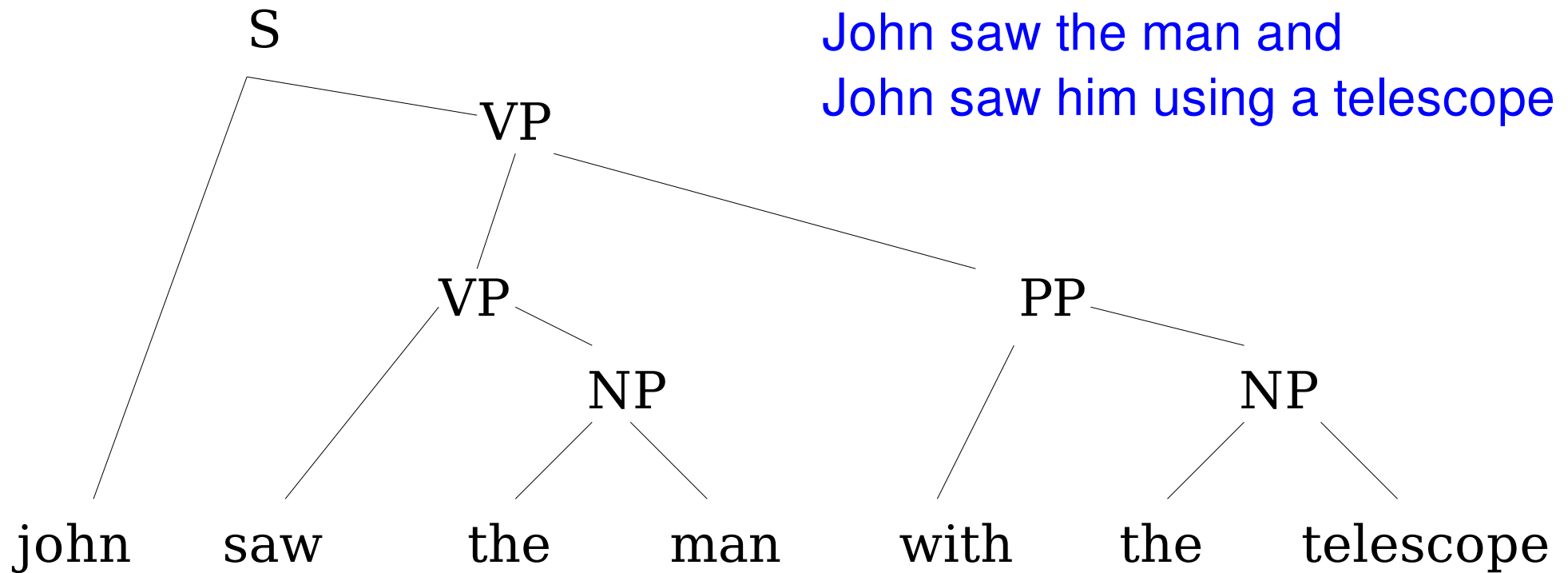
Probabilistic Parsing I



CIS620
Spring, 2005
Ryan McDonald

What is parsing?

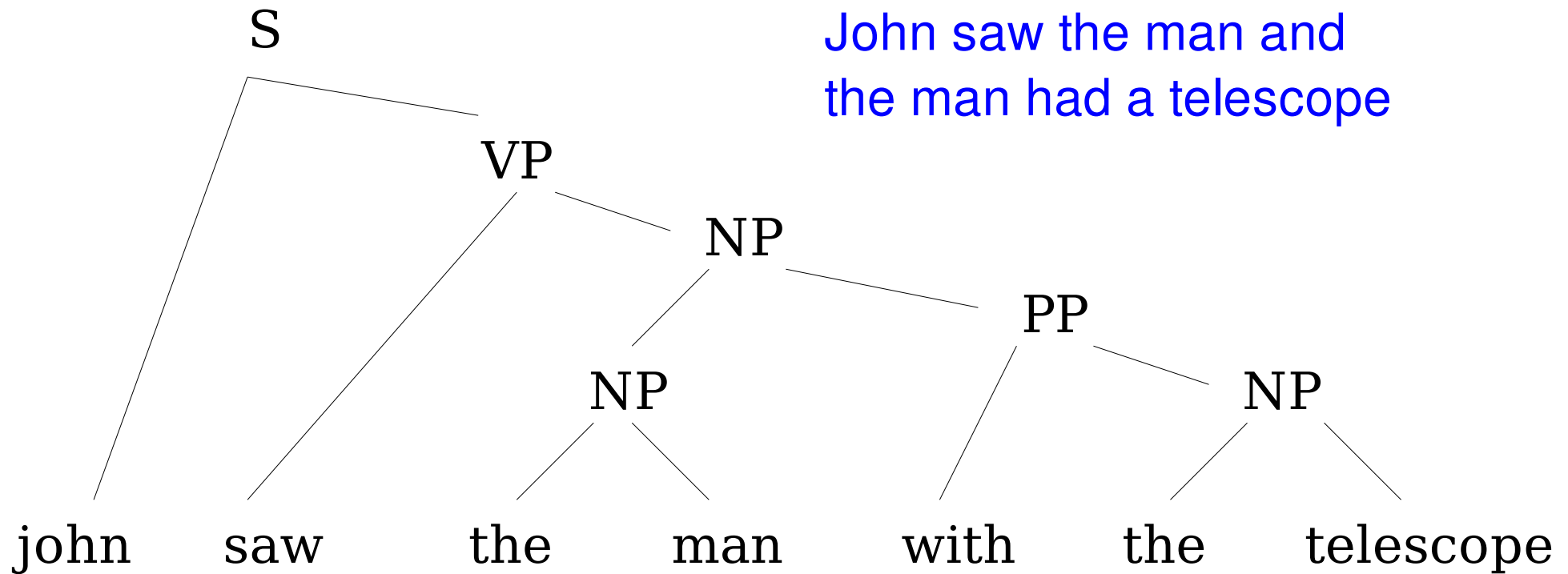
- Input: Sentence, x (String of words)
- Output: Nested Tree, y (a.k.a. Parse Tree)



Parse Tree

What is parsing?

- Input: Sentence, x (String of words)
- Output: Nested Tree, y (a.k.a. Parse Tree)



What is parsing?

- What about?
 - I saw the scientist with the telescope
 - I saw the girl with the telescope
- We can disambiguate, can machines?
- Some questions we will examine:
 - How do we search the space of parse trees?
 - How do we assign probability/scores to parse trees?
 - How do we train these models?
 - Markov assumptions? Viterbi? Forward-Backward?
 - Do these exist for trees?

Why study parsing?

- Parsing is not actually an end itself
 - Parsing important for
 - Information extraction
 - Machine translation
 - Question answering systems
 - Summarization systems
 - etc.
 - Provides a deeper level of syntactic information
 - Above and beyond surface level info (like part-of-speech)
- Parsing is good example of non-sequential structured classification (hot topic in ML)

Schedule

- Today (Feb 15th, 2005)
 - Context-free grammars
 - CKY algorithm
 - PCFGs (generative parsing model)
 - Viterbi and forward-backward (inside-outside) for trees
- Thursday (Feb 17th, 2005)
 - Quick CRF and Perceptron review
 - CRFs/MaxEnt parsing models
 - Perceptron parsing models
 - Open problems and areas of current research

Context-Free Grammars (CFGs)

- We assume that natural language is context-free
 - In particular english
 - Most disagree (but this is way beyond scope)
- A context-free grammar is defined as:

$$G = \{ \Sigma, N, P, S \}$$

where Σ is the alphabet (set of words)

N is the set of nonterminals (tree node labels)

P is a set of productions, $P \subseteq N \times \{N \cup \Sigma\} \times \{N \cup \Sigma\}$

S is a unique start symbol, $S \in N$

Note: binary CFG

$VP \rightarrow saw NP \in P$

$VP \rightarrow saw NP PP \notin P$

! Can always binarize !

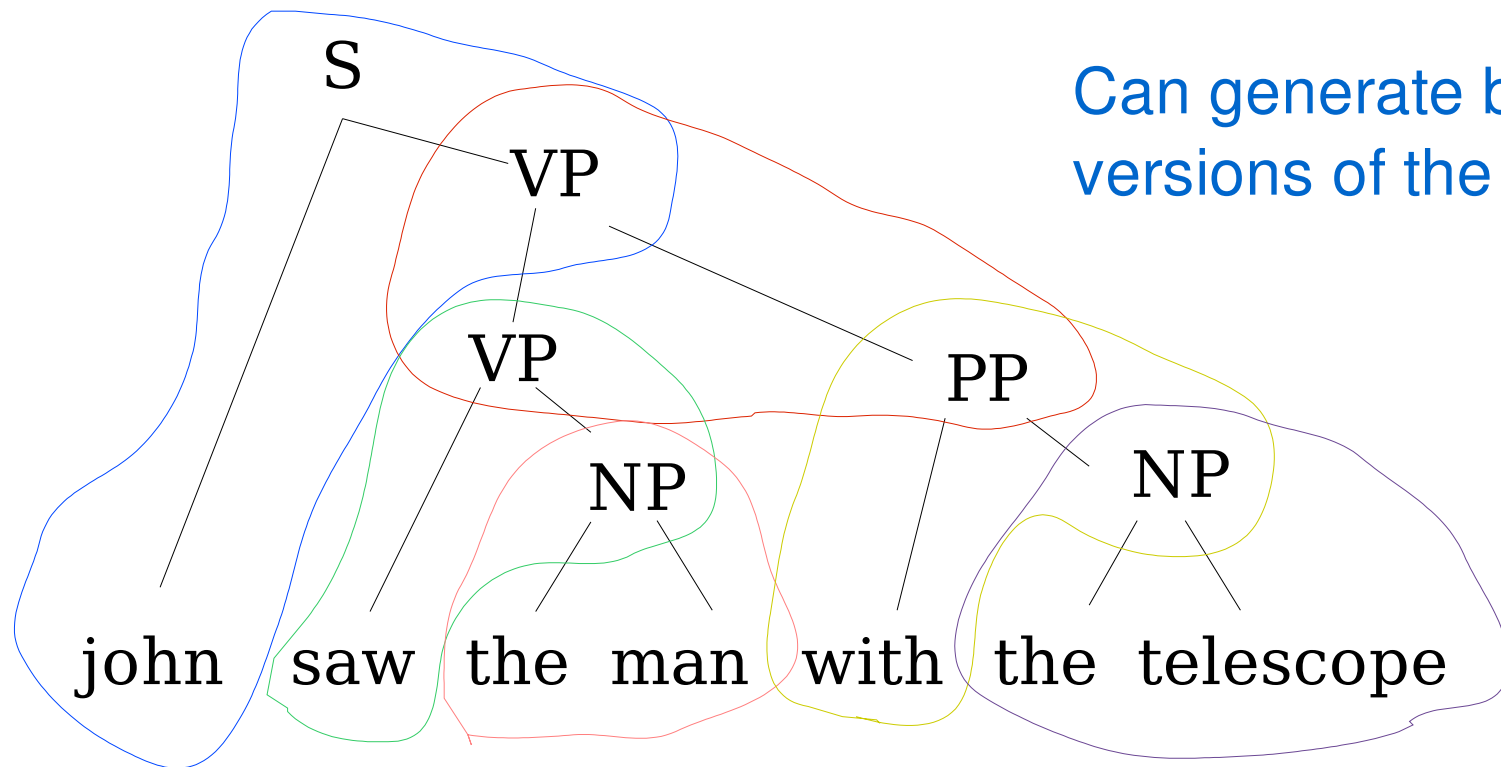
CFG Example

$\Sigma = \{ \text{john, saw, the, man, with, the, telescope} \}$

$N = \{ S, NP, VP, PP \}$

$S = S$

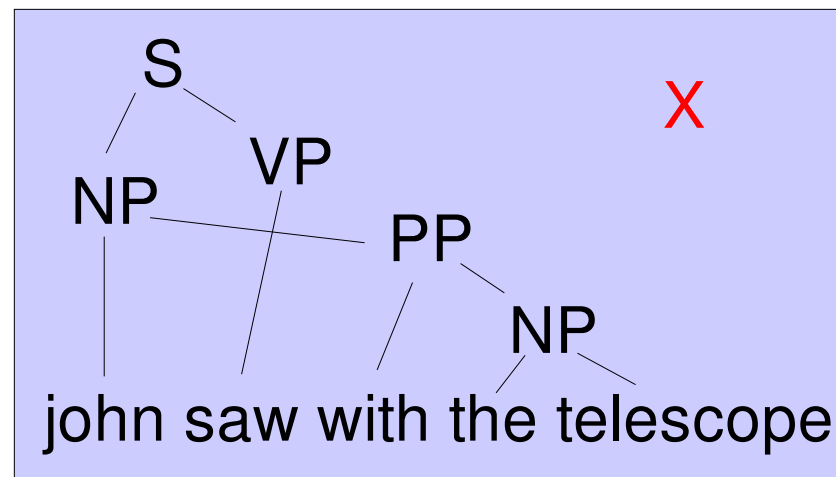
$P = \{ S \rightarrow \text{john VP}, VP \rightarrow VP PP, VP \rightarrow \text{saw NP}, PP \rightarrow \text{with NP}, NP \rightarrow NP PP, NP \rightarrow \text{the man}, NP \rightarrow \text{the telescope} \}$



Can generate both versions of the sentence

CFG: properties

- Some properties of CFGs
 - For an arbitrary sentence, there can be exponentially many valid parse trees (in practice this is also true)
 - Forces nested tree constructions
 - i.e. Does not allow crossings (see below)
 - We will exploit nested property for efficiency



Notation

- For Future reference
- a, b, c, i, j, k – used to index values
- x is always a sentence
 - x_i is the i^{th} word in sentence x
 - x^i is the i^{th} sentence is some set of sentences
- y is always a parse tree
- X, Y, Z are always non-terminal symbols
- $w()$ is a weight/probability function
- w_i is a weight for some feature f_i (next lecture)

Parsing CFGs: CKY

- Question: How do we determine whether a sentence, S , can be derived by some CFG, G ?
- Lots of algorithms: We will use CKY (Cocke, Kasami and Younger)
- Define a dynamic programming table
 - $C[i][j][Z] = \text{true}$
 - Iff it is possible to parse the sentence from word i to word j headed by $Z \in N \cup \Sigma$
 - Hence if $C[1][n][S] = \text{true}$, then we know it is possible to parse the sentence
 - More details ...

CKY

```
Base Case:  $C[i][i][x_i] = \text{true}$   
for  $d = 2 \dots n$   
  for  $i = 1 \dots n - (d - 1)$   
     $j = i + (d - 1)$   
    for  $k = i \dots j - 1$   
      for  $Z \rightarrow X Y \in P$   
        if  $C[i][k][X] \ \&\& \ C[k+1][j][Y]$   
           $C[i][j][Z] = \text{true}$   
        end if  
      end for  
    end for  
  end for  
end for  
Check if  $C[1][n][S] = \text{true}$ 
```

Base Case: all sequences of length 1, x_i is i^{th} word in sentence x

Fill in table with strings of length 2, 3, 4, 5, 6, ..., n

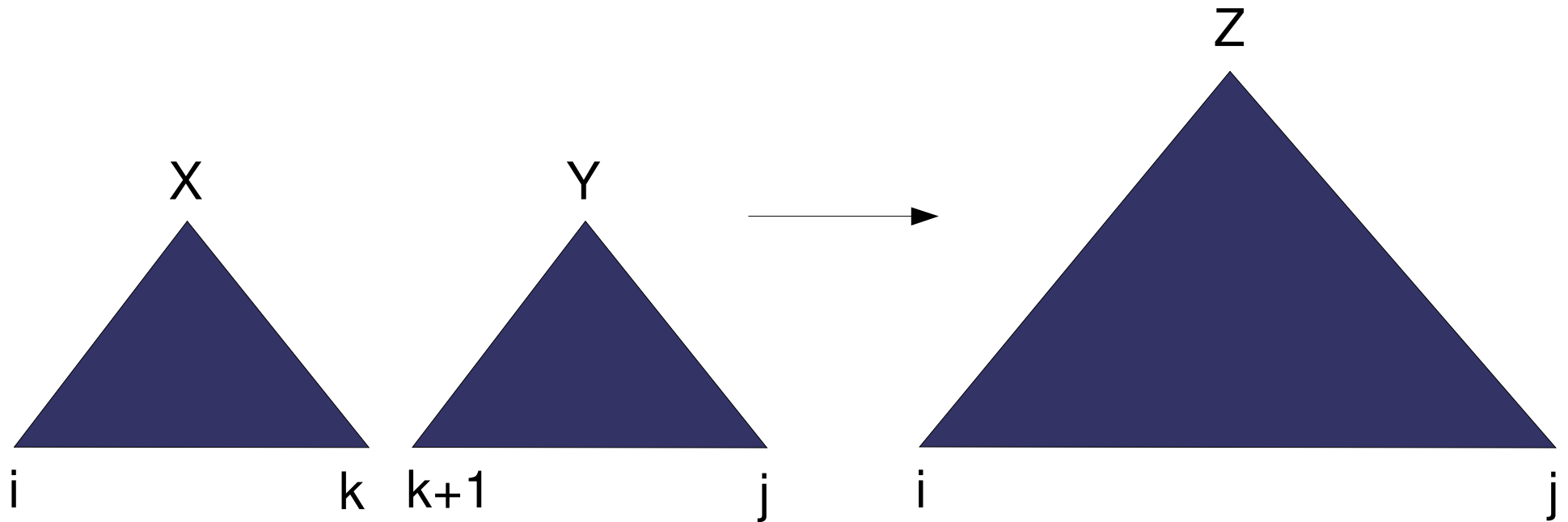
To fill in $C[i][j][*]$, enumerate all smaller items and rules to check for valid combinations

Pretty easy to show this is $O(|P|n^3)$

CKY: graphically

To fill in position $C[i][j][Z]$

- enumerate all $i \leq k < j$ and all $Z \rightarrow X \ Y \in P$
- if valid combo set to true



CKY: example

$\Sigma = \{ \text{john, saw, the, man, with, the, telescope} \}$

$N = \{ S, NP, VP, PP \}$

$S = S$

$P = \{ S \rightarrow \text{john VP}, VP \rightarrow VP PP, VP \rightarrow \text{saw NP}, PP \rightarrow \text{with NP}, NP \rightarrow NP PP, NP \rightarrow \text{the man}, NP \rightarrow \text{the telescope} \}$

john	saw	the	man	with	the	telescope
1	2	3	4	5	6	7

CKY: example

$\Sigma = \{ \text{john, saw, the, man, with, the, telescope} \}$

$N = \{ S, NP, VP, PP \}$

$S = S$

$P = \{ S \rightarrow \text{john VP}, VP \rightarrow VP PP, VP \rightarrow \text{saw NP}, PP \rightarrow \text{with NP}, NP \rightarrow NP PP, NP \rightarrow \text{the man}, NP \rightarrow \text{the telescope} \}$

Base Case

$C[1][1][\text{john}] = \text{true}$

$C[2][2][\text{saw}] = \text{true}$

$C[3][3][\text{the}] = \text{true}$

$C[4][4][\text{man}] = \text{true}$

$C[5][5][\text{with}] = \text{true}$

$C[6][6][\text{the}] = \text{true}$

$C[7][7][\text{telescope}] = \text{true}$

john	saw	the	man	with	the	telescope
1	2	3	4	5	6	7

CKY: example

$\Sigma = \{ \text{john, saw, the, man, with, the, telescope} \}$

$N = \{ S, NP, VP, PP \}$

$S = S$

$P = \{ S \rightarrow \text{john VP}, VP \rightarrow VP PP, VP \rightarrow \text{saw NP}, PP \rightarrow \text{with NP}, NP \rightarrow NP PP, NP \rightarrow \text{the man}, NP \rightarrow \text{the telescope} \}$

Fill in all strings of length 2

$C[1][1][\text{john}] = \text{true}$

$C[2][2][\text{saw}] = \text{true}$

$C[3][3][\text{the}] = \text{true}$

$C[4][4][\text{man}] = \text{true}$

$C[5][5][\text{with}] = \text{true}$

$C[6][6][\text{the}] = \text{true}$

$C[7][7][\text{telescope}] = \text{true}$

$C[3][4][NP] = \text{true}$

Since $C[3][3][\text{the}], C[4][4][\text{man}], NP \rightarrow \text{the man}$

$C[6][7][NP] = \text{true}$

Since $C[6][6][\text{the}], C[7][7][\text{tele}], NP \rightarrow \text{the tele}$

john	saw	the	man	with	the	telescope
1	2	3	4	5	6	7

CKY: example

$\Sigma = \{ \text{john, saw, the, man, with, the, telescope} \}$

$N = \{ S, NP, VP, PP \}$

$S = S$

$P = \{ S \rightarrow \text{john VP}, VP \rightarrow VP PP, VP \rightarrow \text{saw NP}, PP \rightarrow \text{with NP}, NP \rightarrow NP PP, NP \rightarrow \text{the man}, NP \rightarrow \text{the telescope} \}$

Fill in all strings of length 3

$C[1][1][\text{john}] = \text{true}$

$C[2][2][\text{saw}] = \text{true}$

$C[3][3][\text{the}] = \text{true}$

$C[4][4][\text{man}] = \text{true}$

$C[5][5][\text{with}] = \text{true}$

$C[6][6][\text{the}] = \text{true}$

$C[7][7][\text{telescope}] = \text{true}$

$C[3][4][NP] = \text{true}$

$C[6][7][NP] = \text{true}$

$C[2][4][VP] = \text{true}$

Since $C[2][2][\text{saw}], C[3][4][NP], VP \rightarrow \text{saw NP}$

$C[5][7][PP] = \text{true}$

Since $C[5][5][\text{with}], C[6][7][NP], PP \rightarrow \text{with NP}$

john	saw	the	man	with	the	telescope
1	2	3	4	5	6	7

CKY: example

$\Sigma = \{ \text{john, saw, the, man, with, the, telescope} \}$

$N = \{ S, NP, VP, PP \}$

$S = S$

$P = \{ S \rightarrow \text{john VP}, VP \rightarrow VP PP, VP \rightarrow \text{saw NP}, PP \rightarrow \text{with NP}, NP \rightarrow NP PP, NP \rightarrow \text{the man}, NP \rightarrow \text{the telescope} \}$

Continue until done

$C[1][1][\text{john}] = \text{true}$

$C[2][2][\text{saw}] = \text{true}$

$C[3][3][\text{the}] = \text{true}$

$C[4][4][\text{man}] = \text{true}$

$C[5][5][\text{with}] = \text{true}$

$C[6][6][\text{the}] = \text{true}$

$C[7][7][\text{telescope}] = \text{true}$

$C[3][4][NP] = \text{true}$

$C[6][7][NP] = \text{true}$

$C[2][4][VP] = \text{true}$

$C[5][7][PP] = \text{true}$

$C[1][4][S] = \text{true}$

$C[3][7][NP] = \text{true}$

$C[2][7][VP] = \text{true}$

$C[1][7][S] = \text{true}$



Sentence is generated by G

john	saw	the	man	with	the	telescope
1	2	3	4	5	6	7

CFGs con't

- Not all sentences are generated by our simple grammar
 - Generated:
 - John saw the man
 - John saw the telescope
 - John saw the telescope with the man
 - John saw the man with the man
 - ...
 - Not generated:
 - John saw
 - The man saw John with the telescope
 - ...

Simple Example: PCFGs

- Probabilistic-CFGs (PCFGs)
 - PCFGs define a weight/probability for each production rule
 - Denote this weight as $0 \leq w(Z \rightarrow X \ Y) \leq 1$
 - PCFGs define the joint probability of a parse tree and a sentence as:

$$P(x, y) = \prod_{Z \rightarrow X \ Y \in (x, y)} w(Z \rightarrow X \ Y)$$

such that $\forall Z \in N, \sum_{Z \rightarrow X \ Y \in P} w(Z \rightarrow X \ Y) = 1$

Product over all productions use to generate parse tree y for sentence x

PCFGs: properties

- Some properties of PCFGs
 - Given training data $D = \{(x^i, y^i)\}_{i=1..T}$

- The MLE model maximizes

$$\text{MLE}(w) = \arg \max_w \prod_{i=1}^T P(x^i, y^i)$$

$$\text{such that } \forall Z \in N, \sum_{Z \rightarrow X Y, Y \in P} w(Z \rightarrow X Y) = 1$$

- It is easy to show that this results in the following weights

$$w(Z \rightarrow X Y) = \frac{\text{cnt}(Z \rightarrow X Y)}{\sum_{Z \rightarrow X' Y', Y' \in P} \text{cnt}(Z \rightarrow X' Y')}$$

- Where $\text{cnt}(p)$ is the # times production p occurs in D

PCFG Example

$\Sigma = \{ \text{john, saw, the, man, with, the, telescope} \}$

$N = \{ S, NP, VP, PP \}$

$S = S$

$P = \{ S \rightarrow \text{john VP} : 1.0, VP \rightarrow VP PP : 0.4, VP \rightarrow \text{saw NP} : 0.6, PP \rightarrow \text{with NP} : 1.0, NP \rightarrow NP PP : 0.1, NP \rightarrow \text{the man} : 0.6, NP \rightarrow \text{the telescope} : 0.3 \}$

Productions have weights



PCFGs are top-down generative models

john saw the man with the telescope

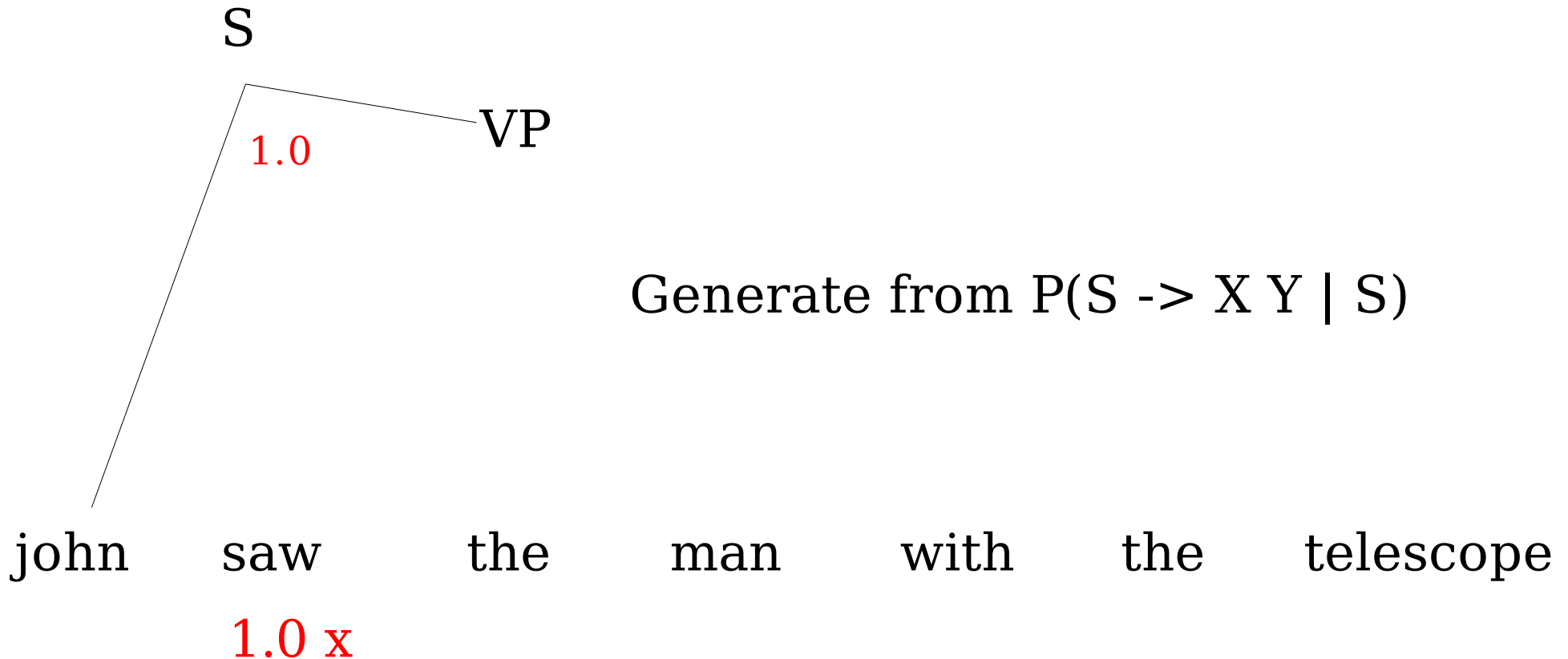
PCFG Example

$\Sigma = \{ \text{john, saw, the, man, with, the, telescope} \}$

$N = \{ S, NP, VP, PP \}$

$S = S$

$P = \{ S \rightarrow \text{john VP} : 1.0, VP \rightarrow \text{VP PP} : 0.4, VP \rightarrow \text{saw NP} : 0.6, PP \rightarrow \text{with NP} : 1.0, NP \rightarrow \text{NP PP} : 0.1, NP \rightarrow \text{the man} : 0.6, NP \rightarrow \text{the telescope} : 0.3 \}$



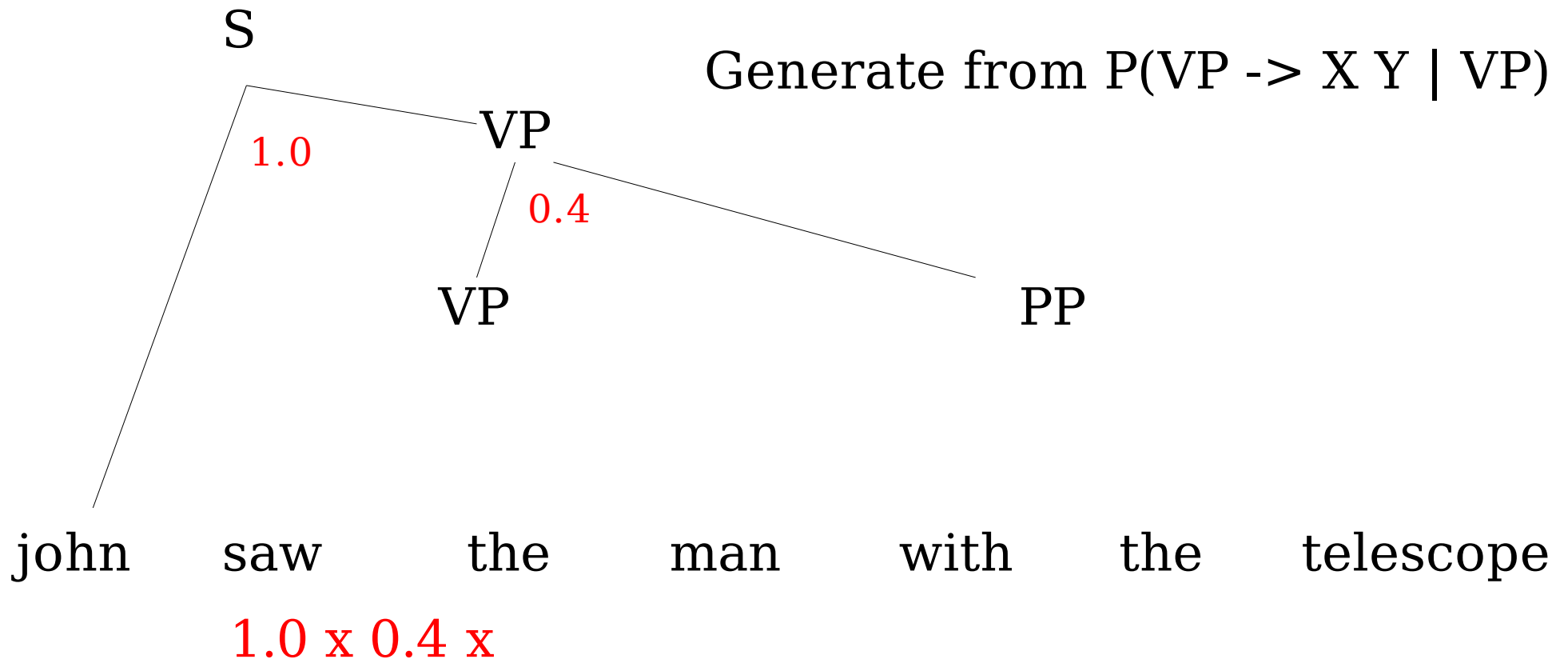
PCFG Example

$\Sigma = \{ \text{john, saw, the, man, with, the, telescope} \}$

$N = \{ S, NP, VP, PP \}$

$S = S$

$P = \{ S \rightarrow \text{john VP} : 1.0, VP \rightarrow VP PP : 0.4, VP \rightarrow \text{saw NP} : 0.6, PP \rightarrow \text{with NP} : 1.0, NP \rightarrow NP PP : 0.1, NP \rightarrow \text{the man} : 0.6, NP \rightarrow \text{the telescope} : 0.3 \}$



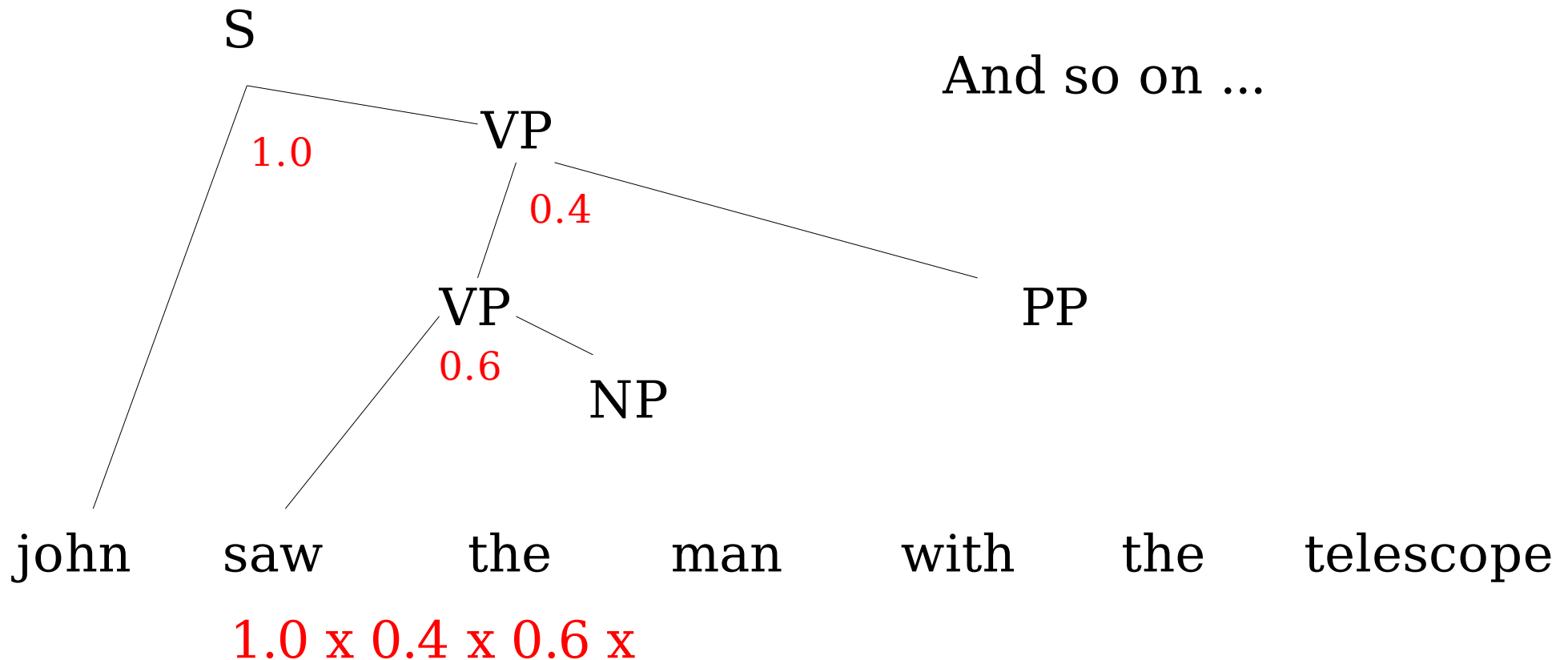
PCFG Example

$\Sigma = \{ \text{john, saw, the, man, with, the, telescope} \}$

$N = \{ S, NP, VP, PP \}$

$S = S$

$P = \{ S \rightarrow \text{john VP} : 1.0, VP \rightarrow VP PP : 0.4, VP \rightarrow \text{saw NP} : 0.6, PP \rightarrow \text{with NP} : 1.0, NP \rightarrow NP PP : 0.1, NP \rightarrow \text{the man} : 0.6, NP \rightarrow \text{the telescope} : 0.3 \}$



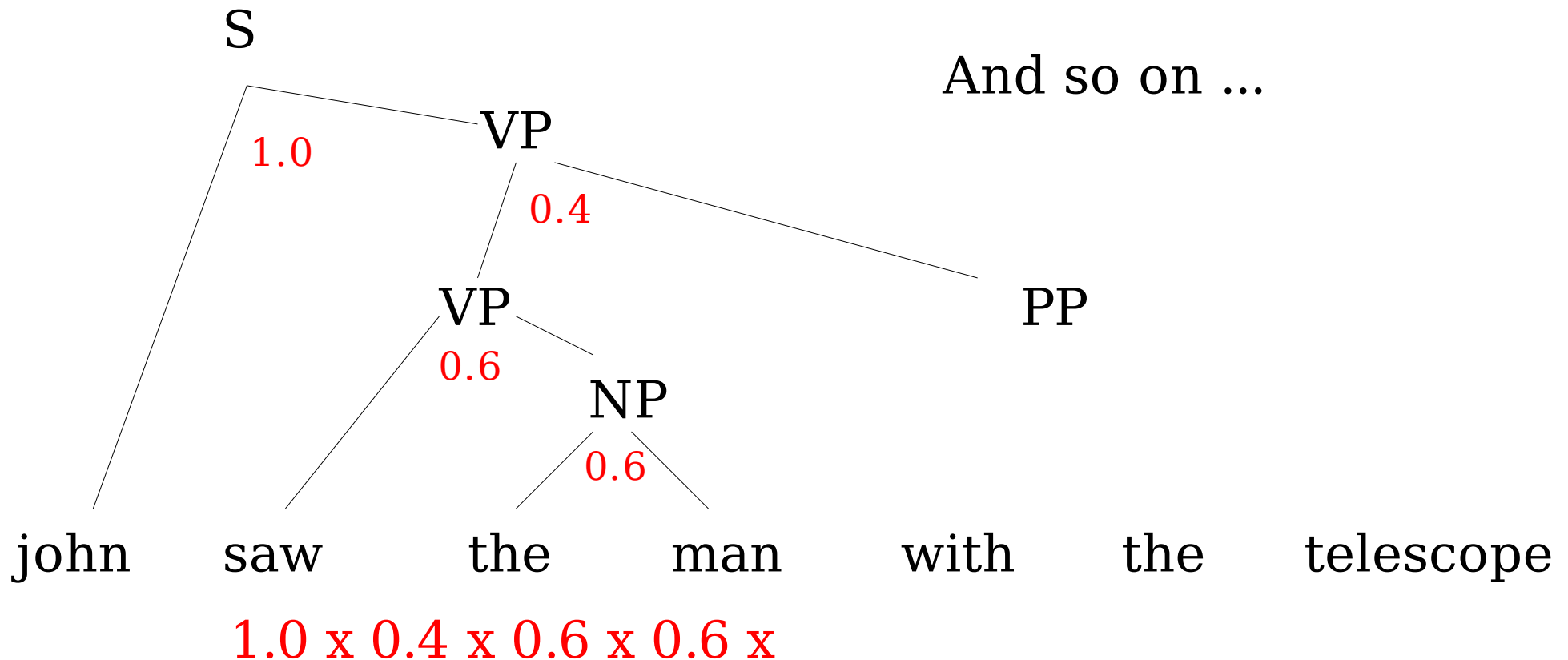
PCFG Example

$\Sigma = \{ \text{john, saw, the, man, with, the, telescope} \}$

$N = \{ S, NP, VP, PP \}$

$S = S$

$P = \{ S \rightarrow \text{john VP} : 1.0, VP \rightarrow VP PP : 0.4, VP \rightarrow \text{saw NP} : 0.6, PP \rightarrow \text{with NP} : 1.0, NP \rightarrow NP PP : 0.1, NP \rightarrow \text{the man} : 0.6, NP \rightarrow \text{the telescope} : 0.3 \}$



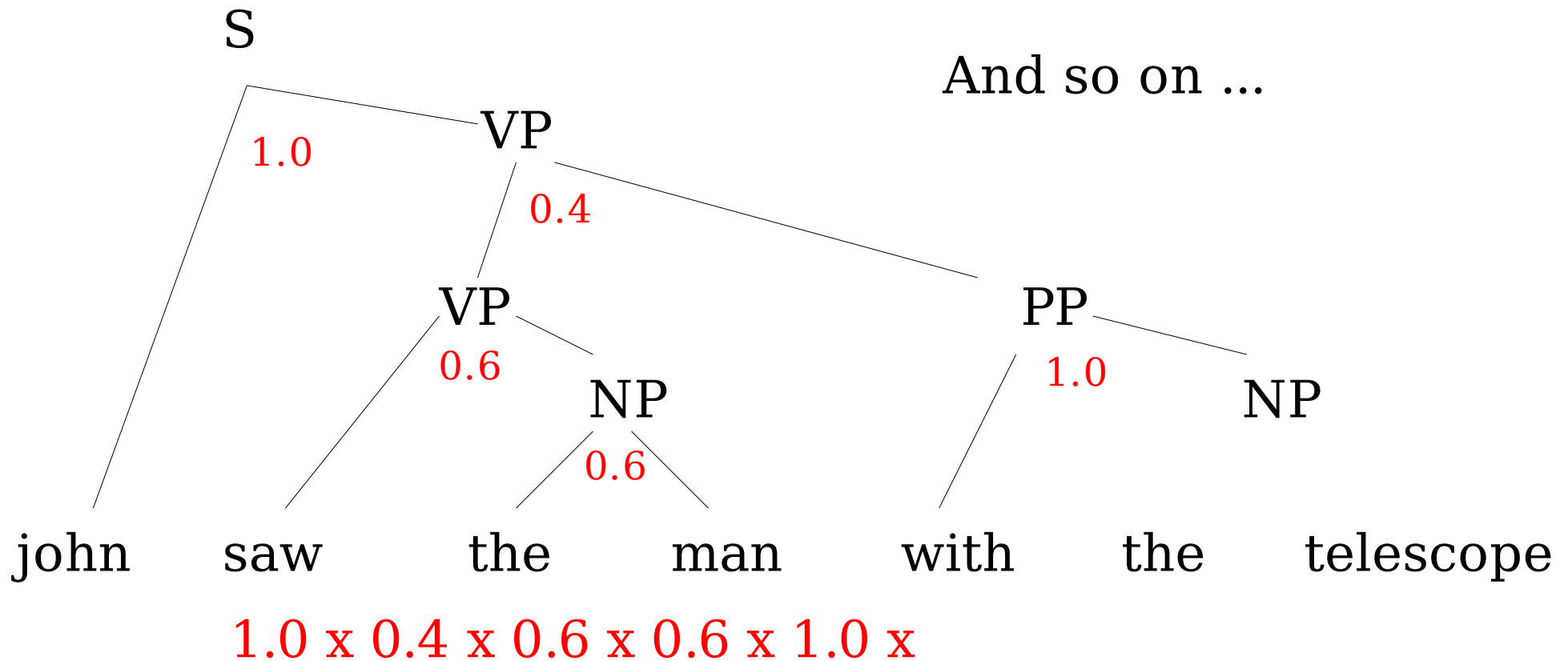
PCFG Example

$\Sigma = \{ \text{john, saw, the, man, with, the, telescope} \}$

$N = \{ S, NP, VP, PP \}$

$S = S$

$P = \{ S \rightarrow \text{john VP} : 1.0, VP \rightarrow VP PP : 0.4, VP \rightarrow \text{saw NP} : 0.6, PP \rightarrow \text{with NP} : 1.0, NP \rightarrow NP PP : 0.1, NP \rightarrow \text{the man} : 0.6, NP \rightarrow \text{the telescope} : 0.3 \}$



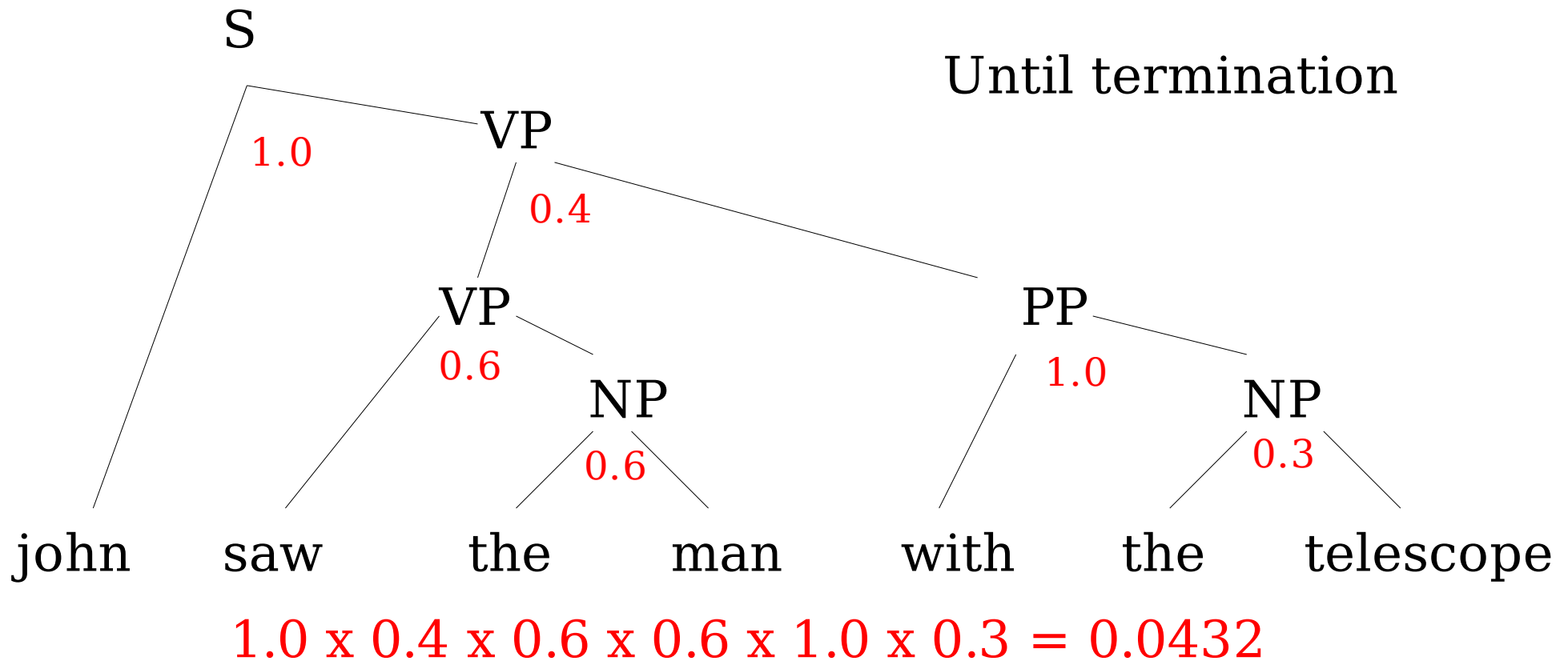
PCFG Example

$\Sigma = \{ \text{john, saw, the, man, with, the, telescope} \}$

$N = \{ S, NP, VP, PP \}$

$S = S$

$P = \{ S \rightarrow \text{john VP} : 1.0, VP \rightarrow \text{VP PP} : 0.4, VP \rightarrow \text{saw NP} : 0.6, PP \rightarrow \text{with NP} : 1.0, NP \rightarrow \text{NP PP} : 0.1, NP \rightarrow \text{the man} : 0.6, NP \rightarrow \text{the telescope} : 0.3 \}$



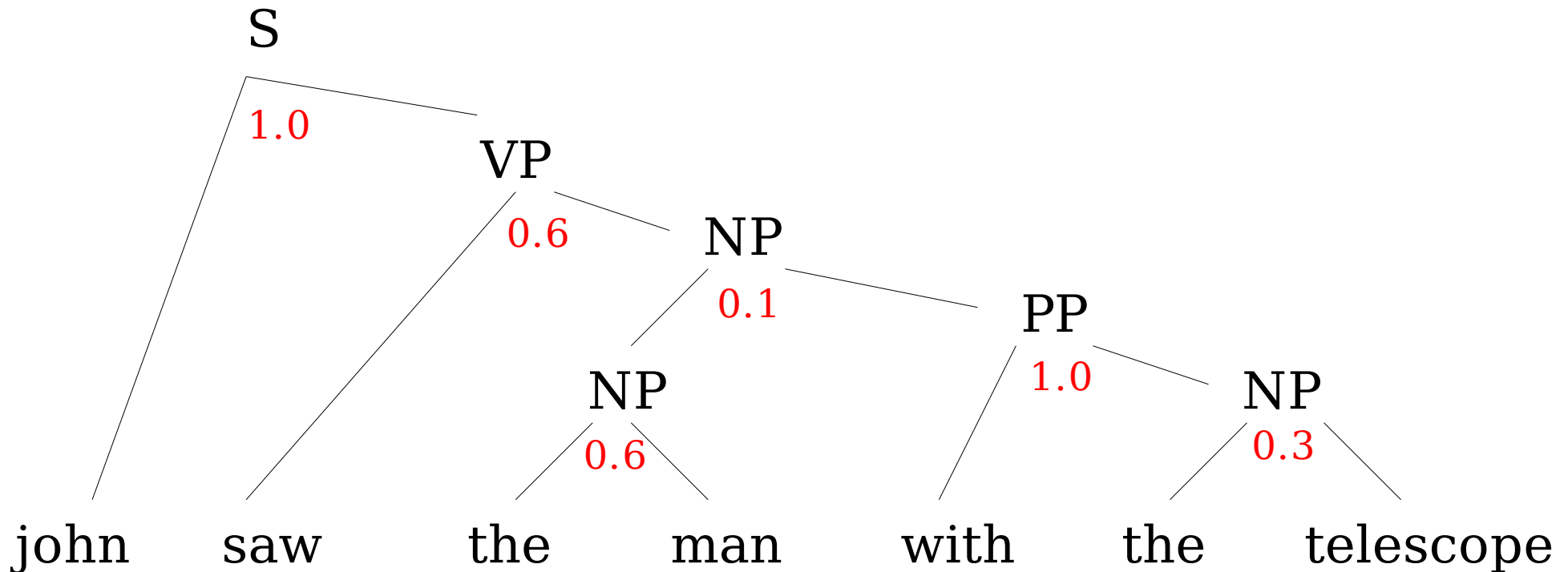
PCFG Example

$\Sigma = \{ \text{john, saw, the, man, with, the, telescope} \}$

$N = \{ S, NP, VP, PP \}$

$S = S$

$P = \{ S \rightarrow \text{john VP} : 1.0, VP \rightarrow \text{VP PP} : 0.4, VP \rightarrow \text{saw NP} : 0.6, PP \rightarrow \text{with NP} : 1.0, NP \rightarrow \text{NP PP} : 0.1, NP \rightarrow \text{the man} : 0.6, NP \rightarrow \text{the telescope} : 0.3 \}$



$$1.0 \times 0.6 \times 0.1 \times 0.6 \times 1.0 \times 0.3 = 0.0108$$

PCFGs: side-note

- Most state-of-the-art parsing models are just PCFGs
 - In particular the Collins model (1999)
- The primary difference is that the best parsing models are lexicalized
 - i.e. Rules are augmented with lexical (word) information
 - Requires slightly different generative process
 - Back-off is very important (actually where power lies)
- Best generative models are about 90% accurate
 - Only measures partial structure accuracy
 - Complete accuracy \ll 50%!!!

Looking back

- Have we addressed the questions?
 - How do we search the space of parse-trees?
 - CKY algorithm
 - How do we model and train tree probabilities?
 - PCFGs
 - Efficient factorization?
 - Every variable/weight is localized to production rules
 - Just another Markov assumption
 - Are there analogous algorithms for Viterbi decoding and forward-backward calculations?
 - Yes. Both can easily be generalized to tree structures.
 - Next ...

PCFGs: Viterbi Decoding

- Given a PCFG, G and a sentence, x
 - How do we find the best parse-tree?

$$y = \operatorname{argmax}_y P(y | x) = \operatorname{argmax}_y \frac{P(x, y)}{P(x)} = \operatorname{argmax}_y P(x, y)$$

- Can modify CKY to store best parses
 - $W[i][j][Z]$ = weight of best tree from i to j rooted at Z
 - $C[i][j][Z] = \text{false}$ then $W[i][j][Z] = 0.0$
 - $B[i][j][Z]$ = back pointer to items that made best parse
 - $B[i][j][Z] = \{(i, k, X), (k+1, j, Y)\}$

PCFGs: Viterbi Decoding

Base Case: $C[i][i][x_i] = \text{true}$, $W[*][*][*] = 0.0$, $W[i][i][x_i] = 1.0$

for $d = 2 \dots n$

 for $i = 1 \dots n-(d-1)$

$j = i+(d-1)$

 for $k = i \dots j - 1$

 for $Z \rightarrow X \ Y \in P$

 if $C[i][k][X] \ \&\& \ C[k+1][j][Y]$

$C[i][j][Z] = \text{true}$

$v = W[i][k][X] \times W[k+1][j][Y] \times w(Z \rightarrow X \ Y)$

 if $v > W[i][j][Z]$ ←

$W[i][j][Z] = v$

$B[i][j][Z] = \{(i,k,X), (k+1,j,Y)\}$

 end if

 end if

 end for

 end for

 end for

end for

Follow back pointers from $B[1][n][S]$ to get best parse

If this parse has a higher probability then the current max, update max values and back pointers

PCFGs: Viterbi Decoding

Base Case: $C[i][i][x_i] = \text{true}$, $W[*][*][*] = 0.0$, $W[i][i][x_i] = 1.0$

for $d = 2 \dots n$

 for $i = 1 \dots n-(d-1)$

$j = i+(d-1)$

 for $k = i \dots j - 1$

 for $Z \rightarrow X \ Y \in P$

 if $C[i][k][X] \ \&\& \ C[k+1][j][Y]$

$C[i][j][Z] = \text{true}$

$v = W[i][k][X] \times W[k+1][j][Y] \times w(Z \rightarrow X \ Y)$

 if $v > W[i][j][Z]$

$W[i][j][Z] = v$

$B[i][j][Z] = \{(i,k,X), (k+1,j,Y)\}$

 end if

 end if

 end for

 end for

 end for

end for

Follow back pointers from $B[1][n][S]$ to get best parse

Best way to understand algorithm is to try it out on a simple sentence and grammar

PCFGs: Forward-Backward

- Is it possible to calculate forward-backward probabilities?
 - Yes. But for trees we call them inside-outside probs
- First we will look at calculation inside probabilities
- Forward/Inside probabilities
 - Before: $\alpha_t(s)$
 - Probability of seeing x_1, \dots, x_t given we are in state s
 - Now: $\alpha(i, j, Z)$
 - Probability of seeing sub-tree from i to j rooted by Z

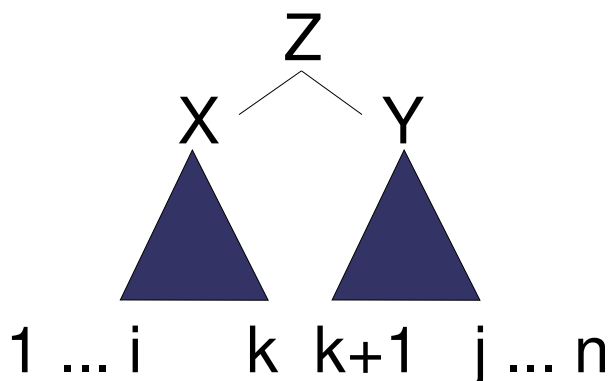
PCFGs: Inside probabilities

- Inside probability: $\alpha(i, j, Z)$
 - Probability of seeing sub-tree from i to j rooted by Z

$$\alpha(i, i, x_i) = 1.0, \quad \alpha(i, i, Z) = 0.0, \quad \forall Z \in N \cup \Sigma, \text{ s.t. } Z \neq x_i$$

$$\alpha(i, j, Z) = \sum_{i \leq k < j, Z \rightarrow X \ Y \in P} \alpha(i, k, X) \times \alpha(k+1, j, Y) \times w(Z \rightarrow X \ Y)$$

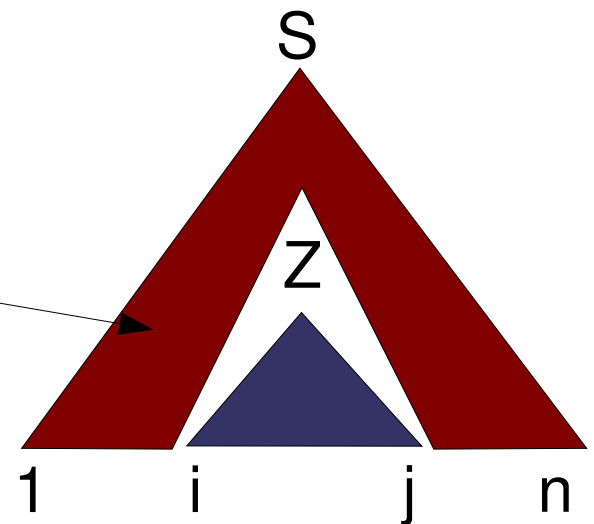
- Viterbi/CKY is easily modified to calculate inside probs



PCFGs: Outside probabilities

- Outside probabilities are a little trickier
- Backward/outside probabilities
 - Before: $\beta_t(s)$
 - Probability of seeing x_t, \dots, x_n given we are in state s
 - Now: $\beta(i, j, Z)$
 - Probability of completing tree with a constituent from i to j rooted by Z

Want to calculate this probability mass



PCFGs: Outside probabilities

- Calculating outside probabilities

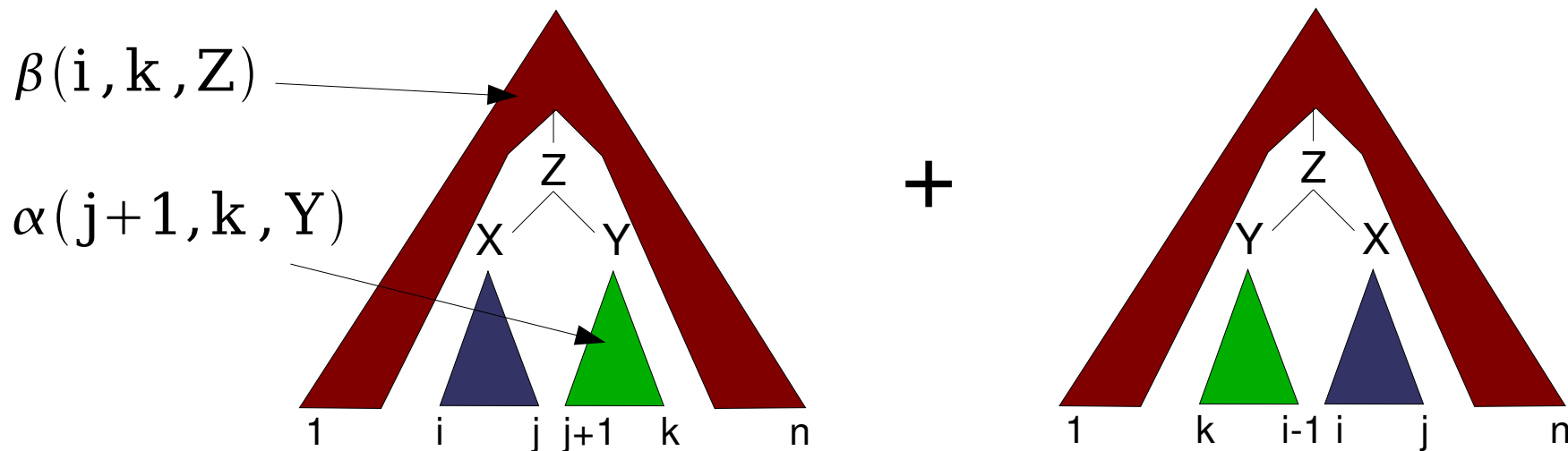
$$\beta(1, n, S) = 1.0$$

$$\beta(1, n, Z) = 0.0 \quad \forall Z \neq S$$

$$\beta(i, j, X) =$$

$$\sum_{j < k \leq n, Z \rightarrow X \ Y \in P} \beta(i, k, Z) \times \alpha(j+1, k, Y) \times w(Z \rightarrow X \ Y)$$

$$+ \sum_{1 \leq k < i, Z \rightarrow Y \ X \in P} \beta(k, j, Z) \times \alpha(k, i-1, X) \times w(Z \rightarrow Y \ X)$$



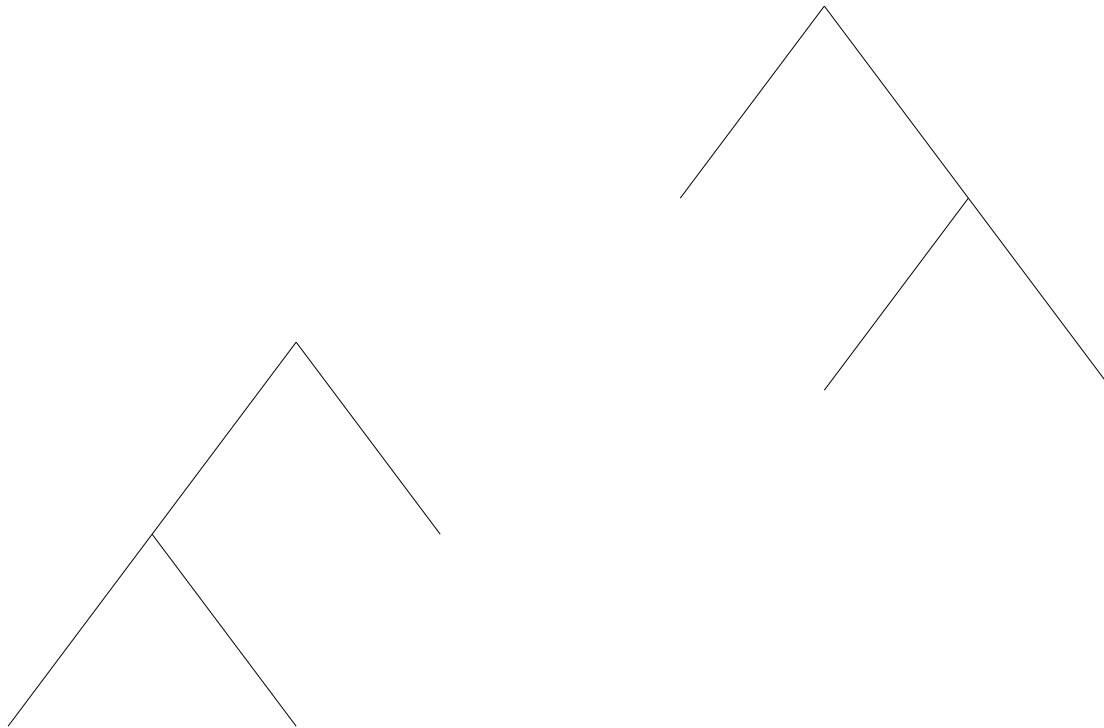
Inside-outside probabilities con't

- Can modify CKY/Viterbi to also calculate outside probs
 - Requires top-down dynamic programming
 - i.e. First do all sequences of length n , then $n-1$, ..., then 1
- CKY is backbone of all PCFG calculations
 - Hence complexity is $O(|P|n^3)$ for everything
- This is problematic since:
 - Average sentence length is 23, with sentences commonly being longer than 40 words
 - $|P|$ can be in the thousands (and tens of thousands)
 - Efficient parsing has been an active area of research

Why do we need I-O probabilities?

- We can already train PCFGs and do Viterbi
- But i-o probabilities often come in handy
 - Unlabeled data: Can train EM algorithm with parse trees as hidden variables (called the inside-outside algorithm)
 - Just like the HMM case
 - Usually results in poor parses
 - Conditional random field parsing
 - Forward-backward scores used to calculate feature expectations in sequential case
 - Will do exactly this for parsing using i-o probs
 - Next time ...

Probabilistic Parsing II



CIS620
Spring, 2005
Ryan McDonald

Outline

- What we covered so far
 - CFGs, CKY, PCFGs, Viterbi and inside-outside
- This time: Discriminative models
 - Feature based exponential and linear models
 - Constraints on features
 - Conditional Random Fields for parsing
 - Perceptron parsing
 - Issues with discriminative parsing
- Extra: State-of-the-art = Lexicalization & Re-ranking

Feature based models

- Have seen evidence before that rich dependent feature sets improve performance
 - HMM vs. MEMM vs. CRF
- Dependent features are intractable in generative models
 - Solution: only model conditional distribution
 - Define discriminative probabilities/scores

$$\text{CRFs: } P(y|x) = \frac{e^{\sum_a w_a f_a(x,y)}}{Z(x)}$$

$f_a(x, y)$ = Some feature over a sentence, x , and parse tree, y

$$\text{SVMs \& Perceptron: } \text{score}(x, y) = \sum_a w_a f_a(x, y)$$

Feature based models

- What sort of features are allowed?
 - In order to maintain polynomial complexity we force features to be over productions
 - In particular over information we can obtain during the CKY algorithm
 - This will allow us to use Viterbi, inside-outside, etc.

$f_a(x,y) = 1.0$ iff y contains VP \rightarrow saw NP

$f_a(x,y) = 1.0$ iff y contains VP \rightarrow * NP

$f_a(x,y) = 1.0$ iff y contains VP \rightarrow saw NP and NP contains "man"

$f_a(x,y) = 1.0$ iff y contains S \rightarrow John VP and VP is headed by "saw"

etc.

* is wild card

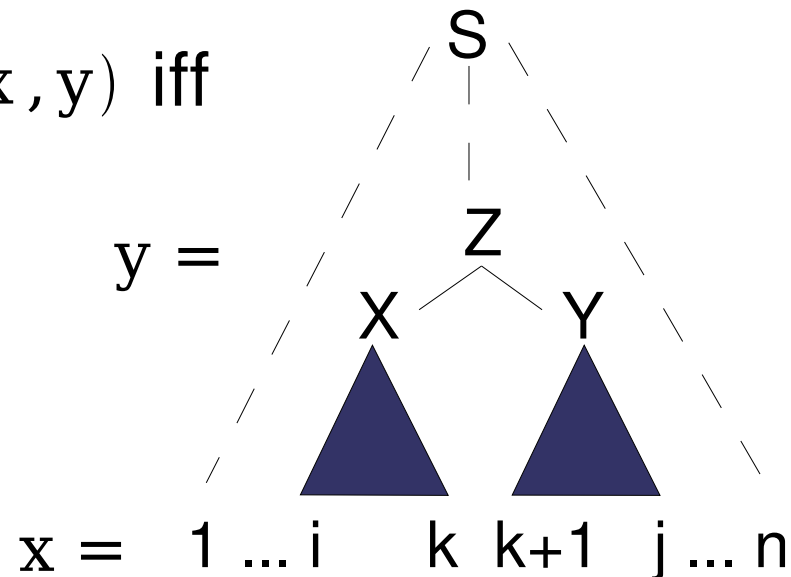
For lexicalized CFGs (more later)

Feature based models

- Hence we can re-write:

$$\sum_a w_a f_a(\mathbf{x}, \mathbf{y}) = \sum_{(Z \rightarrow X Y, i, k, j) \in (\mathbf{x}, \mathbf{y})} \sum_a w_a f_a(Z \rightarrow X Y, i, k, j)$$

- Where $(Z \rightarrow X Y, i, k, j) \in (\mathbf{x}, \mathbf{y})$ iff



Why is this constraint necessary?

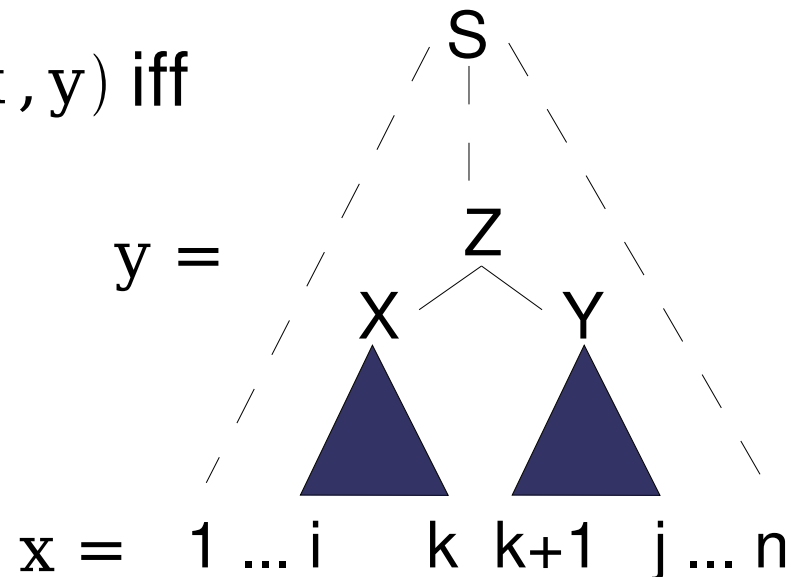
Feature based models

- Hence we can re-write:

$$\sum_a w_a f_a(\mathbf{x}, \mathbf{y}) = \sum_{(Z \rightarrow X Y, i, k, j) \in (\mathbf{x}, \mathbf{y})} \sum_a w_a f_a(Z \rightarrow X Y, i, k, j)$$

- Where $(Z \rightarrow X Y, i, k, j) \in (\mathbf{x}, \mathbf{y})$ iff

Because our dynamic programming algorithms (CKY) only maintain information over productions and spans. Hence to run Viterbi and i-o we must force all calculations to be restricted to productions.



Feature based models: Viterbi

- Lets assume that we have a set of weights, w_a
- How do we find the best tree given a sentence x
- CRFs and linear models use identical argmax functions

$$y = \arg \max_y P(y|x) = \arg \max_y \frac{e^{\sum_a w_a f_a(x, y)}}{Z(x)}$$
$$= \arg \max_y e^{\sum_a w_a f_a(x, y)} = \arg \max_y \sum_a w_a f_a(x, y)$$

- Can we change CKY/Viterbi to find best y ?
 - Yes, we will now just sum scores (not multiply probs)

PCFGs: Viterbi Decoding

Base Case: $C[i][i][x_i] = \text{true}$, $W[*][*][*] = -\text{infinity}$, $W[i][i][x_i] = 0.0$

for $d = 2 \dots n$

 for $i = 1 \dots n-(d-1)$

$j = i+(d-1)$

 for $k = i \dots j - 1$

 for $Z \rightarrow X \ Y \in P$

 if $C[i][k][X] \ \&\& \ C[k+1][j][Y]$

$C[i][j][Z] = \text{true}$

$v = W[i][k][X] + W[k+1][j][Y] + \sum_a w_a f_a(Z \rightarrow X \ Y, i, k, j)$

 if $v > W[i][j][Z]$

$W[i][j][Z] = v$

$B[i][j][Z] = \{(i, k, X), (k+1, j, Y)\}$

 end if

 end if

 end for

 end for

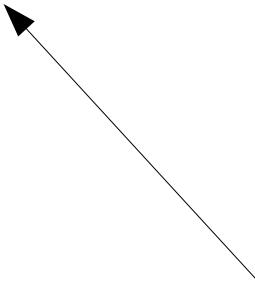
 end for

end for

Start from 0.0 since additive



Score of a constituent is score of sub-trees plus score of adding this rule to the tree from i to j



CRFs/linear models: Viterbi

Base Case: $C[i][i][x_i] = \text{true}$, $W[*][*][*] = -\text{infinity}$, $W[i][i][x_i] = 0.0$

for $d = 2 \dots n$

 for $i = 1 \dots n-(d-1)$

$j = i+(d-1)$

 for $k = i \dots j - 1$

 for $Z \rightarrow X \ Y \in P$

 if $C[i][k][X] \ \&\& \ C[k+1][j][Y]$

$C[i][j][Z] = \text{true}$

$v = W[i][k][X] + W[k+1][j][Y] + \sum_a w_a f_a(Z \rightarrow X \ Y, i, k, j)$

 if $v > W[i][j][Z]$

$W[i][j][Z] = v$

$B[i][j][Z] = \{(i, k, X), (k+1, j, Y)\}$

 end if

 end if

 end for

 end for

 end for

end for

By restricting our features to production rules it allows use to use CKY/Viterbi



Conditional Random Fields: Review

- Last week Fernando talked about CRFs for sequences
- However, CRFs are more general algorithms
 - Can be formulated on any undirected graph using clique factorization
 - Some graphs are intractable
 - Alternate view: can be run on any problem in which feature expectations can be calculated
- Side note: CRF parsing is often called MaxEnt parsing in the NLP community
 - Not sure why, since we are training global models based on structural factorization

Conditional Random Field: Review

CRFs model the conditional probability of a structure, y , given some input, x , as a normalized exponential feature based model

$$P(y|x) = \frac{e^{\sum_a w_a f_a(x, y)}}{Z(x)}, \quad Z(x) = \sum_{y'} e^{\sum_a w_a f_a(x, y')}$$

CRFs are discriminative/conditional models that maximize the conditional likelihood, $L(w)$, of the training data

$$\begin{aligned} w &= \arg \max_w L(w) = \arg \max_w \sum_{t=1}^{|T|} \log P(y^t | x^t) \\ &= \arg \max_w \sum_{t=1}^{|T|} \log \frac{e^{\sum_a w_a f_a(x^t, y^t)}}{Z(x^t)} \end{aligned}$$

Training data

$T = (x^t, y^t)_{t=1 \dots |T|}$

Conditional Random Fields: Review

We know from previous lectures that there is no closed form solution to this maximization. Hence we need to use either iterative scaling algorithms:

$$\text{GIS: } w_a \leftarrow w_a + \frac{1}{C} \log \left(\frac{\tilde{\mathbf{E}}[f_a | \mathbf{T}]}{\mathbf{E}[f_a | \mathbf{T}]} \right)$$

See logistic regression slides

Or gradient based algorithms (gradient ascent, conjugate gradient, limited memory quasi-Newton):

$$\frac{\delta L(\mathbf{w})}{\delta w_a} = \tilde{\mathbf{E}}[f_a | \mathbf{T}] - \mathbf{E}[f_a | \mathbf{T}] \quad w_a \leftarrow w_a + \eta \left(\frac{\delta L(\mathbf{w})}{\delta w_a} \right)$$

Empirical feature expectations

$$\tilde{\mathbf{E}}[f_a | \mathbf{T}] = \sum_{t=1}^{|\mathbf{T}|} f_a(\mathbf{x}^t, \mathbf{y}^t)$$

Model feature expectations

$$\mathbf{E}[f_a | \mathbf{T}] = \sum_{t=1}^{|\mathbf{T}|} \sum_y P(\mathbf{y} | \mathbf{x}^t) f_a(\mathbf{x}^t, \mathbf{y})$$

Conditional Random Fields: Review

- So to train a CRF model, we need:

- Empirical feature expectations: $\tilde{E}[f_a|\mathbf{T}] = \sum_{t=1}^{|\mathbf{T}|} f_a(\mathbf{x}^t, \mathbf{y}^t)$

- Easy to calculate, just counts over training data

- Model feature expectations: $E[f_a|\mathbf{T}] = \sum_{t=1}^{|\mathbf{T}|} \sum_y P(\mathbf{y}|\mathbf{x}^t) f_a(\mathbf{x}^t, \mathbf{y})$

- For MaxEnt/Logistic Regression this is easy

- But structured data typically has exponentially many labelings, \mathbf{y} , for a given input, \mathbf{x}

- Hence CRFs factorize labelings using Markov assumptions

- Can calculate expectations for sequences using forward-backward

- We will calculate feature expectations for trees using inside-outside probabilities

Model expectations for trees

- Lets do some re-writting:

$$E[f_a|T] = \sum_{t=1}^{|T|} E[f_a|x^t] = \sum_{t=1}^{|T|} \sum_{Z \rightarrow X Y \in P} \sum_{1 \leq i \leq k < j \leq n} E[Z \rightarrow X Y, i, k, j | x^t] f_a(Z \rightarrow X Y, i, k, j)$$

Features are restricted to productions
 - Expectation of a feature is equals to expectation of a production over some span times by the value of the feature

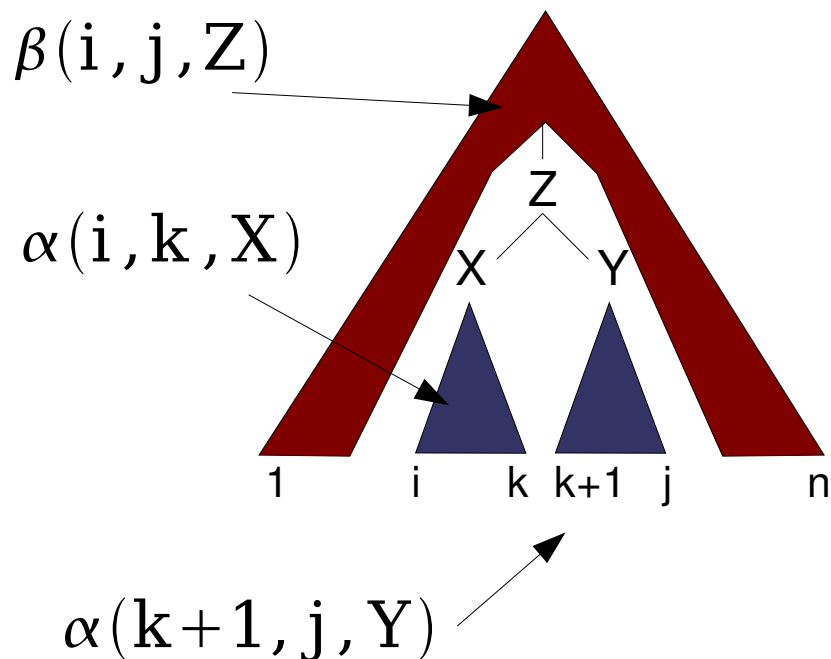
- So what we really need to calculate for sentence x^t is:

$$E[Z \rightarrow X Y, i, k, j | x^t]$$

- For all $Z \rightarrow X Y \in P, 1 \leq i \leq k < j \leq n$ ← Length of x^t

Model expectations for trees

$$\begin{aligned}
 E[Z \rightarrow X \ Y, i, k, j | \mathbf{x}^t] &= \sum_{y, \text{ s.t. } (Z \rightarrow X \ Y, i, k, j) \in (\mathbf{x}^t, y)} P(y | \mathbf{x}^t) \\
 &= \frac{\alpha(i, k, X) \times \alpha(k+1, j, Y) \times \beta(i, j, Z) \times e^{\sum_a w_a f_a(Z \rightarrow X \ Y, i, k, j)}}{Z(\mathbf{x})}
 \end{aligned}$$



$$Z(\mathbf{x}) = \alpha(1, n, S)$$

The expectation is equal to the SCORE of all trees subtrees times the score of completing the tree times the score of including this rule – all normalized

Model expectations for trees

- Note that the values we need are analogous to i-o probabilities – except that they are scores and not probabilities
 - i.e. Exponentiated linear sums

$$\alpha(i, i, x_i) = 1.0$$

$$\alpha(i, j, Z) = \sum_{Z \rightarrow X \ Y, i \leq k < j} \alpha(i, k, X) \times \alpha(k+1, j, Y) \times e^{\sum_a w_a f_a(Z \rightarrow X \ Y, i, k, j)}$$

$$\beta(1, n, S) = 1.0$$

$$\beta(1, n, Z) = 0.0 \quad \forall Z \neq S$$

$$\beta(i, j, X) =$$

$$\sum_{j < k \leq n, Z \rightarrow X \ Y \in P} \beta(i, k, Z) \times \alpha(j+1, k, Y) \times e^{\sum_a w_a f_a(Z \rightarrow X \ Y, i, j, k)}$$
$$+ \sum_{1 \leq k < i, Z \rightarrow Y \ X \in P} \beta(k, j, Z) \times \alpha(k, i-1, Y) \times e^{\sum_a w_a f_a(Z \rightarrow X \ Y, k, i, j)}$$

Can calculate with CKY/Viterbi!!

Model expectations for trees

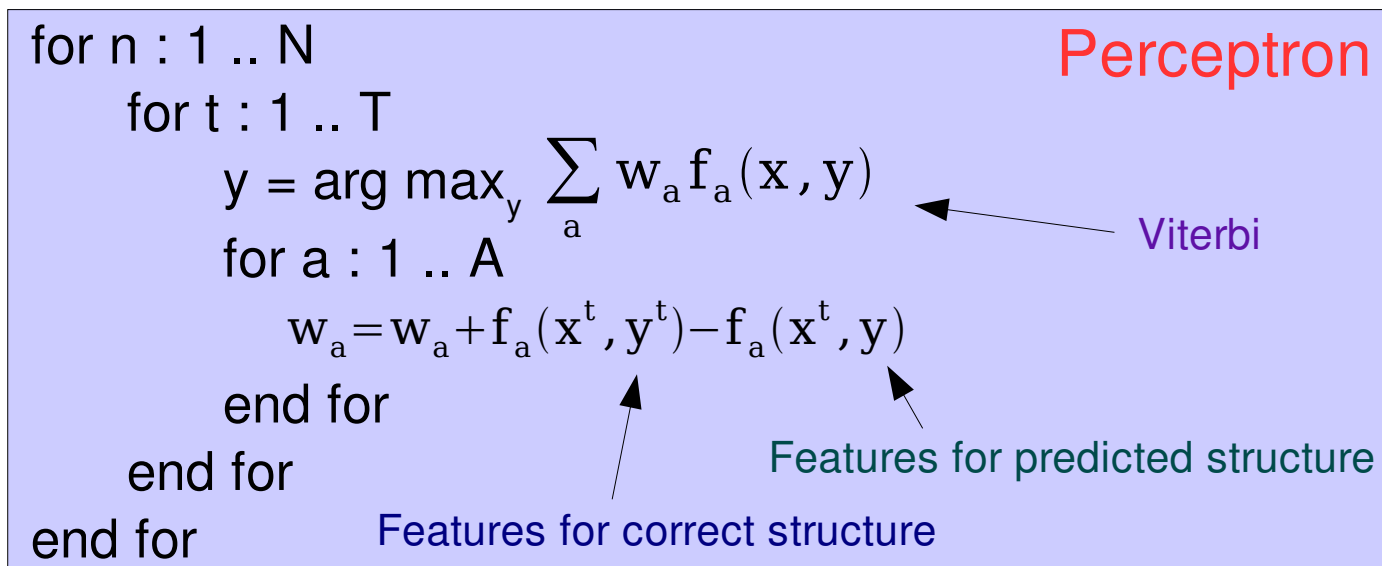
- Can calculate i-o scores in $O(|P|n^3)$ using CKY
 - Hence, can calculate $E[Z \rightarrow X Y, i, k, j | x^t]$ in $O(1)$ if i-o scores known
 - So to calculate $E[f_a | T]$
 - For each training instance
 - Calculate i-o scores
 - Then $\sum_{Z \rightarrow X Y \in P} \sum_{1 \leq i \leq k < j \leq n} E[Z \rightarrow X Y, i, k, j | x^t] f_a(Z \rightarrow X Y, i, k, j)$
 - Add result to total
 - Whole training procedure is run in $O(|T||P|n^3)$
-

Parsing with CRFs Summary

- In order to train a CRF model we need to calculate feature expectations
 - Can do this using efficient factorization and i-o scores
 - Whole procedure takes $O(|T||P|n^3)$
 - But we have to do this every training iteration!
 - $|T| = 40,000$
 - $|P|$ on order of thousands
 - n on average is around 23
- Once model is trained, can use CKY/Viterbi to find best parse

Perceptron parsing

- Perceptron for structured data?
 - Perceptron style algorithms actually very powerful
 - They are discriminative models since they attempt to minimize prediction error
 - Collins (2002) extended them to general structured outputs
 - We already have all tools necessary: CKY/Viterbi



Perceptron parsing

- Perceptron for structured data?
 - Perceptron style algorithms actually very powerful
 - Discriminative since they attempt to minimize prediction error
 - Collins (2002) extended them to general structured outputs
 - We already have all tools necessary: Viterbi

for n : 1 .. N
for t : 1 .. T
y = arg max_y $\sum_{(Z \rightarrow X Y, i, k, j) \in (x^t, y)}$ $\sum_a w_a f_a(Z \rightarrow X Y, i, k, j)$
for a : 1 .. A
w_a = w_a + [$\sum_{(Z \rightarrow X Y, i, k, j) \in (x^t, y^t)}$ f_a(Z → X Y, i, k, j)] - [$\sum_{(Z \rightarrow X Y, i, k, j) \in (x^t, y)}$ f_a(Z → X Y, i, k, j)]
end for
end for
end for

Perceptron Parsing

↑ Features for correct parse ↑ Features for predicted parse

Perceptron parsing

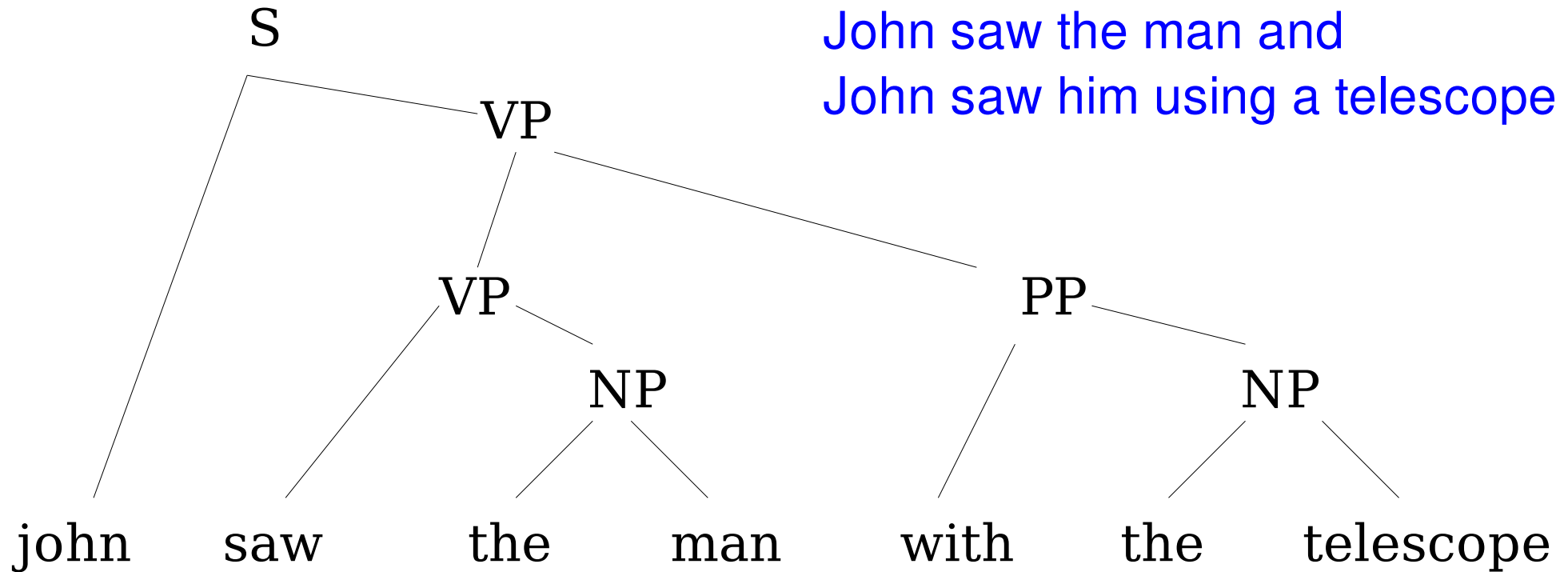
- Perceptron is nice since
 - It is simple, easy to understand and implement
- But ...
 - Each training iteration is also $O(|T||P|n^3)$ since we need to do Viterbi/CKY – just as bad as CRFs
 - But we can often approximate Viterbi search without much harm (pruning). Approx i-o usually very detrimental.
 - CRFs usually outperform all variants of perceptron

END OF DISCRIMINATIVE PARSING

- Up next, current trends: lexicalization and re-ranking

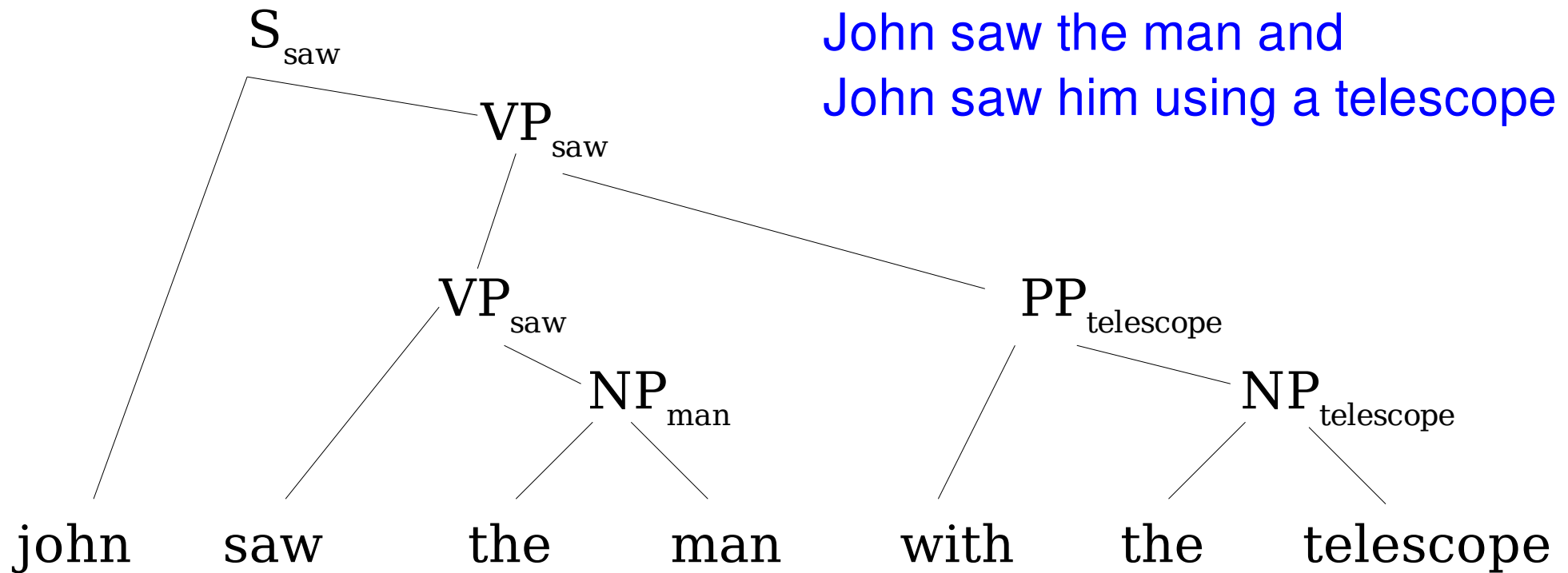
State-of-the-art parsing

- We have talked only about basic CFG parsing
 - The best parsers are "lexicalized"



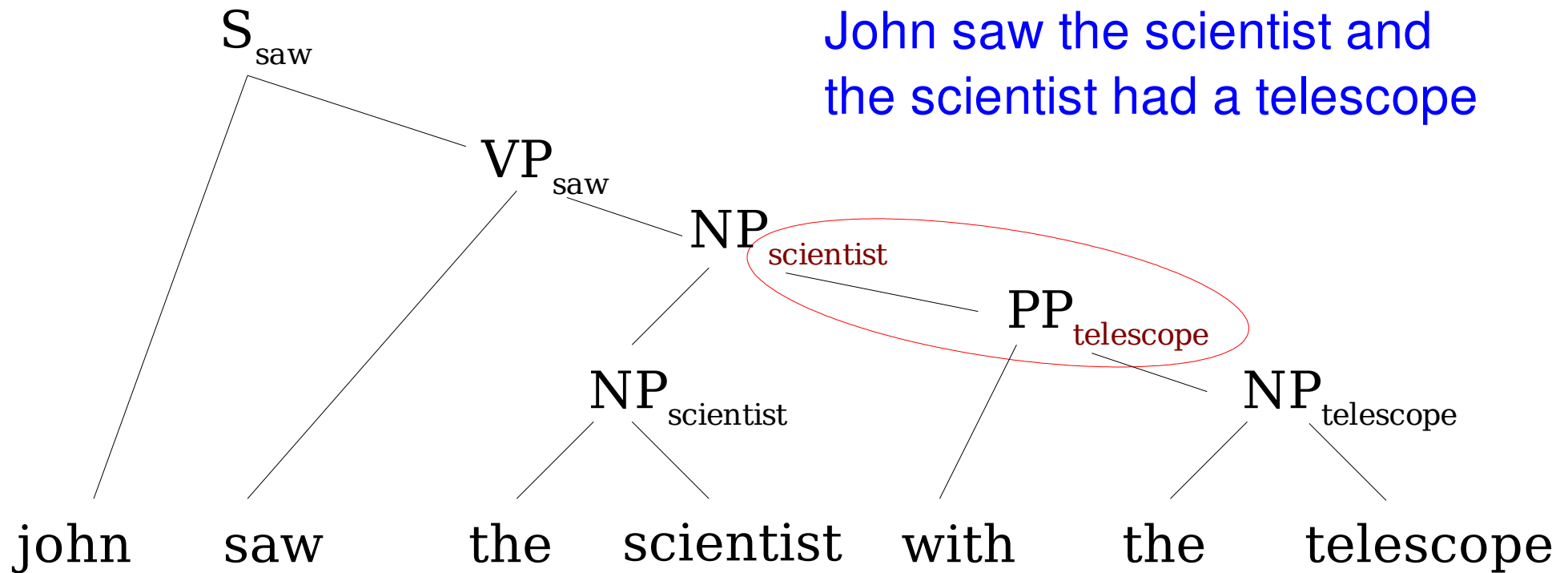
State-of-the-art parsing

- We have talked only about basic CFG parsing
 - The best parsers are "lexicalized"



State-of-the-art parsing

- Why is lexicalized parsing a good thing to do?
 - Allows us to calculate stats over correlated words

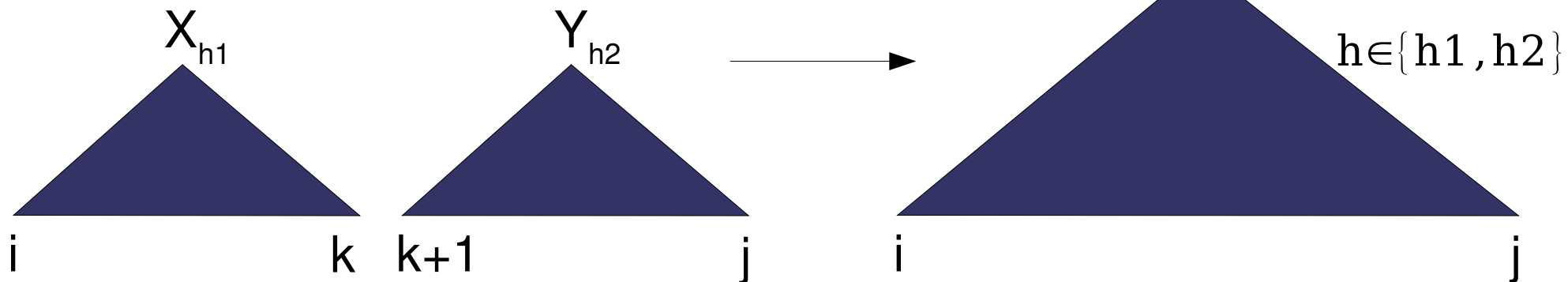


Lexicalization

- How do we lexicalize a parse-tree?
 - Usually follow some fixed set of rules
 - Use EM
- How does this change parsing? Increases to $O(|P|n^5)$

To fill in position $C[i][j][h][Z]$

- enumerate all $i \leq k < j$, $i \leq h_1 \leq k$,
 $k+1 \leq h_2 \leq j$ and all $Z \rightarrow X Y \in P$



Lexicalization

- What about our parsing models?

- PCFGs:
$$w(Z_h \rightarrow X_{h1} Y_{h2}) = \frac{\text{cnt}(Z_h \rightarrow X_{h1} Y_{h2})}{\sum_{Z_h \rightarrow X'_{h1}, Y'_{h2}} \text{cnt}(Z_h \rightarrow X'_{h1}, Y'_{h2})}$$

- Results in extremely sparse distributions

- Must use back-off

$$w(Z_h \rightarrow X_{h1} Y_{h2}) = \lambda_1 w(Z_h \rightarrow X_{h1} Y_{h2}) + \lambda_2 w(Z \rightarrow X Y) + \lambda_3 w(h \rightarrow h1 h2) + \dots$$

- Train back-off with deleted interpolation, Katz smoothing, ...

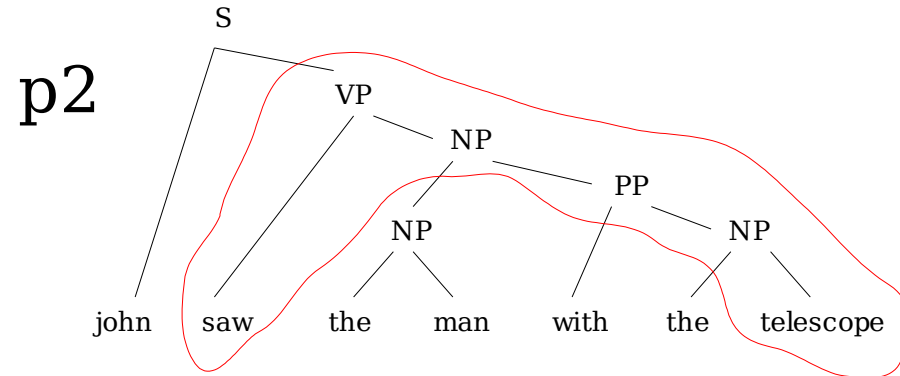
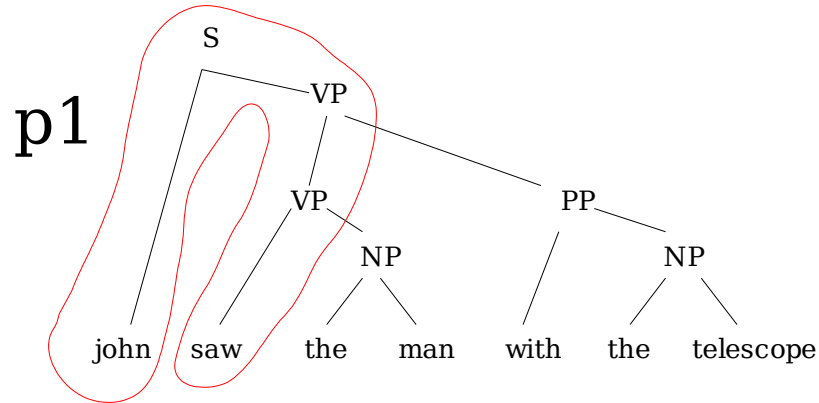
- Designing back-off models has really been the heart of modern parsing

Re-ranking

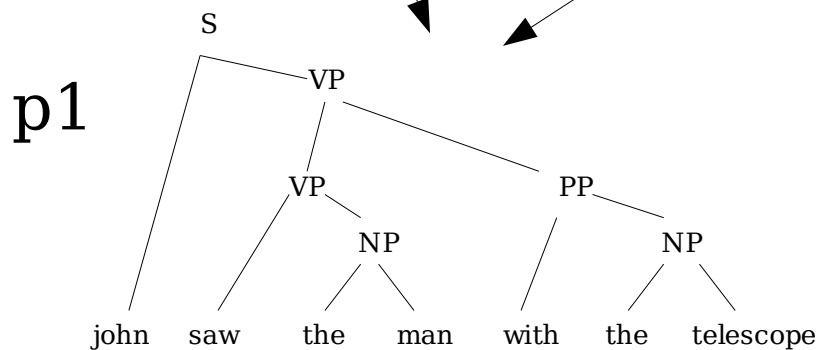
- Problems with discriminative parsing
 - $O(|T||P|n^3)$ training complexity, $O(|T||P|n^5)$ lexicalized
 - Currently just cannot run models of this size
 - People have tried, but most require heavy pruning and other approximations
- One solution:
 - Take k-best parses from generative model
 - Use a discriminative model to re-rank them
 - Turns problem into multi-class classification
 - Allows features over entire tree structure

Re-ranking

- Say $k = 2$ and a PCFG spits out



$$P(y \mid x)$$



Model $P(y \mid x)$, where y in $\{p1, p2\}$
Can use arbitrary feature over entire structure

Since we don't need to calculate Viterbi or i-o
Don't need to search entire space, just k different parses

Re-ranking

- Lots of algorithms: perceptron, ada-boost, SVMs
 - Some algorithms use tree kernels
- Generally re-ranking results in a 10% decrease in error
- Generating the k-best parses is an active area of research
 - Better k-best distributions = better re-ranking
- Maybe some day we will be able to train full discriminative models without re-ranking
 - All discriminative models today either prune or only work for very short sentences

Review

- Basics of parsing
 - CFGs, CKY, PCFGs, Viterbi and inside-outside algorithm
- Discriminative models
 - CRFs: calculation expectations with i-o scores
 - Perceptron: simple, all we need is Viterbi
 - Drawback: $O(|T||P|n^3)$ training
 - Solution – re-ranking
- Lexicalization
 - Allows us to gather meaningful word-word statistics