Training PCFGs

Computational Linguistics

Alexander Koller

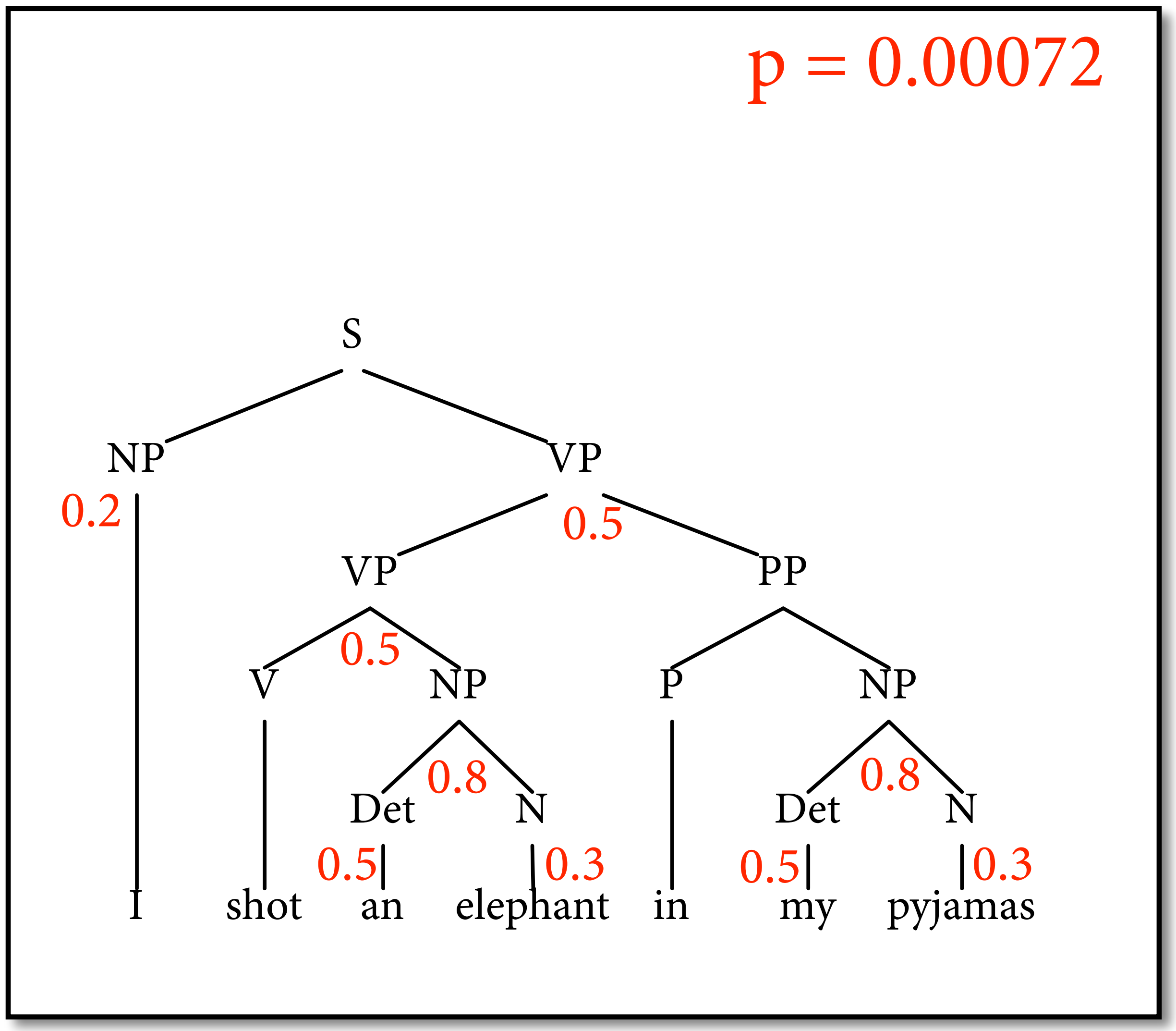
30 November 2018

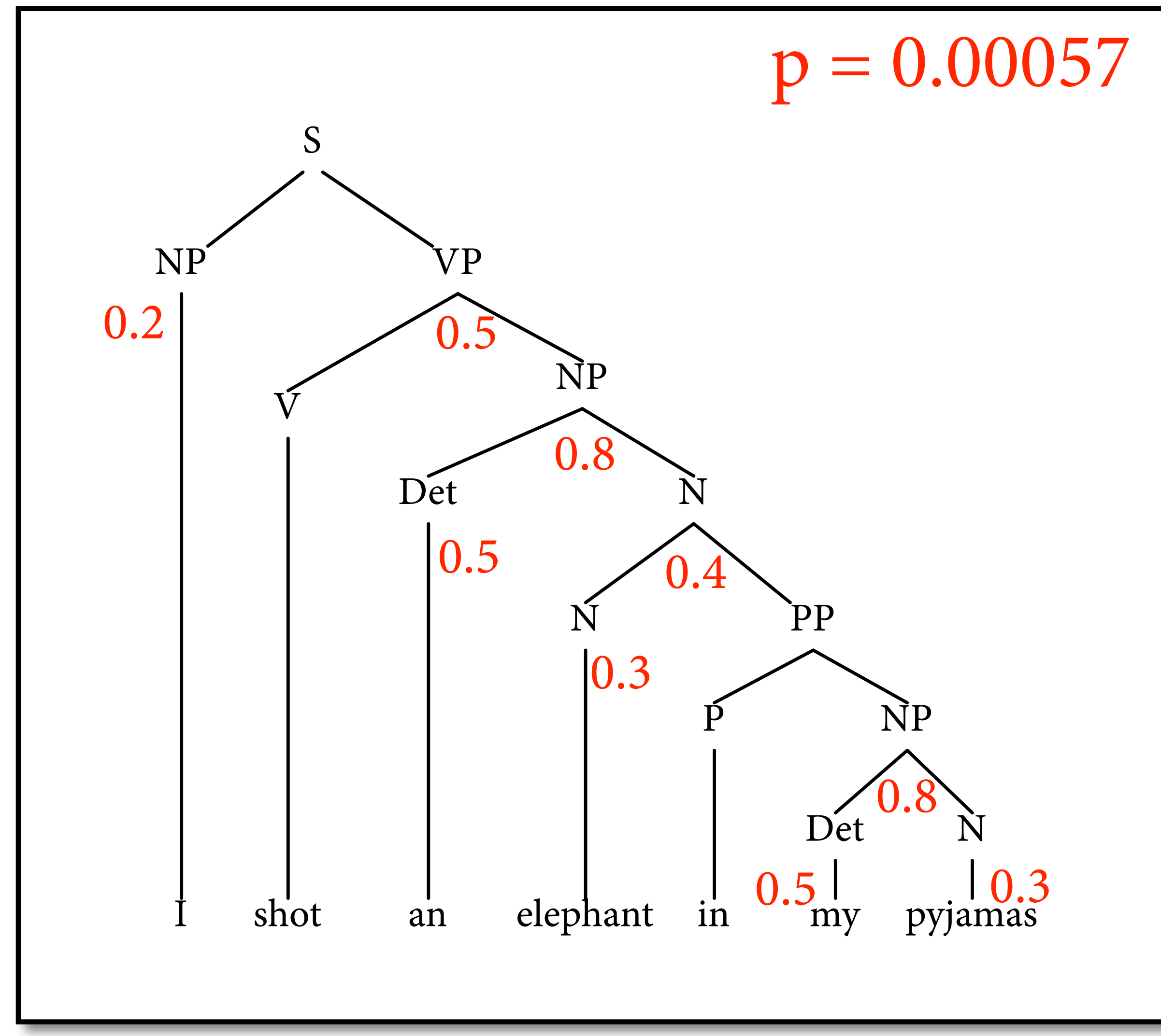
Probabilistic CFGs

S → NP VP	[1.0]	VP → V NP	[0.5]
NP → Det N	[0.8]	VP → VP PP	[0.5]
NP → i	[0.2]	V → shot	[1.0]
N → N PP	[0.4]	PP → P NP	[1.0]
N → elephant	[0.3]	P → in	[1.0]
N → pyjamas	[0.3]	Det → an	[0.5]
		Det → my	[0.5]

(let's pretend for simplicity that Det = PRP$)

Parse trees

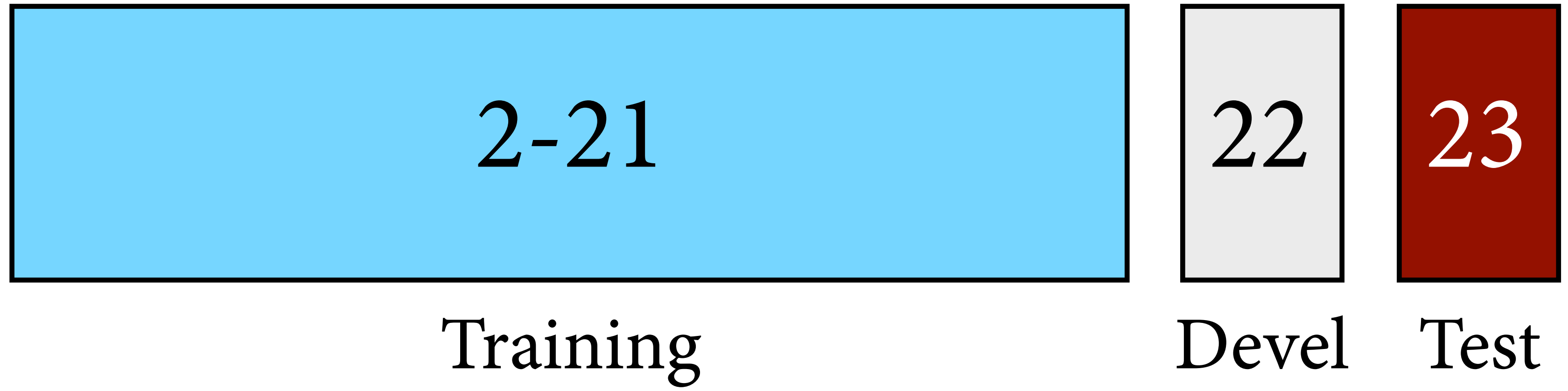




"correct" = more probable parse tree

Evaluation

- Step 1: Decide on training and test corpus.
  For WSJ corpus, there is a conventional split by sections:

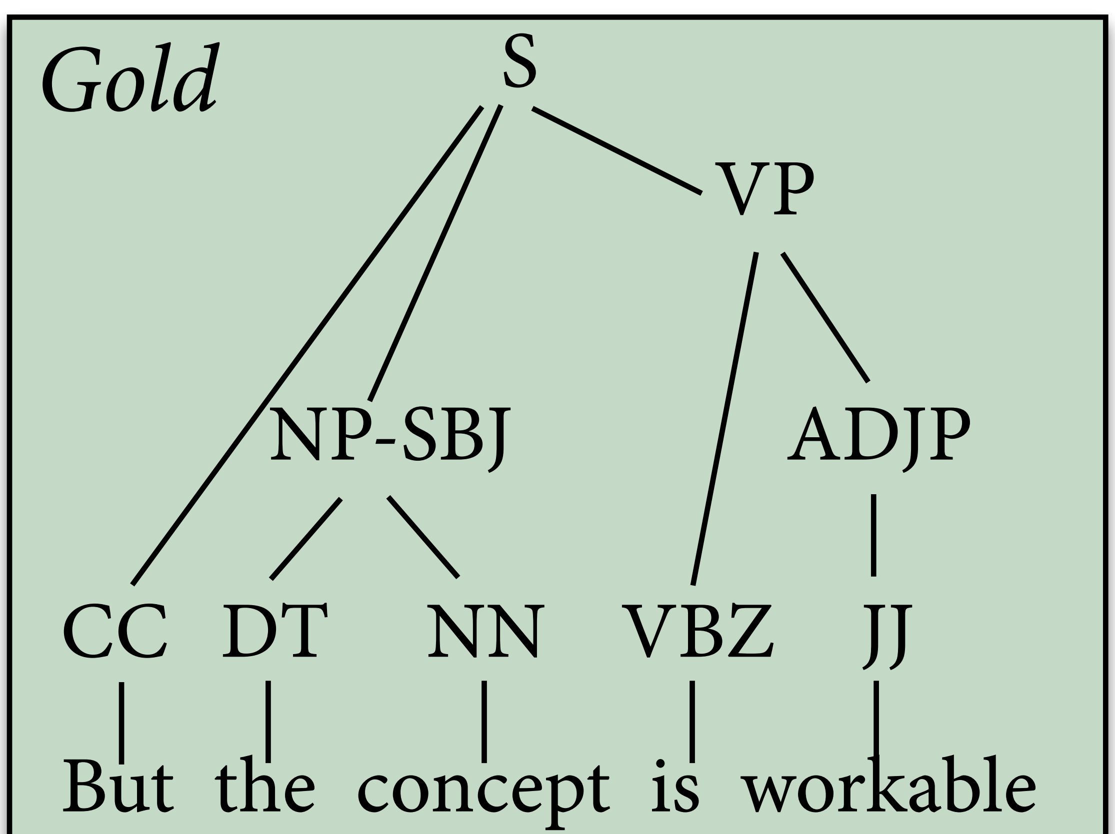


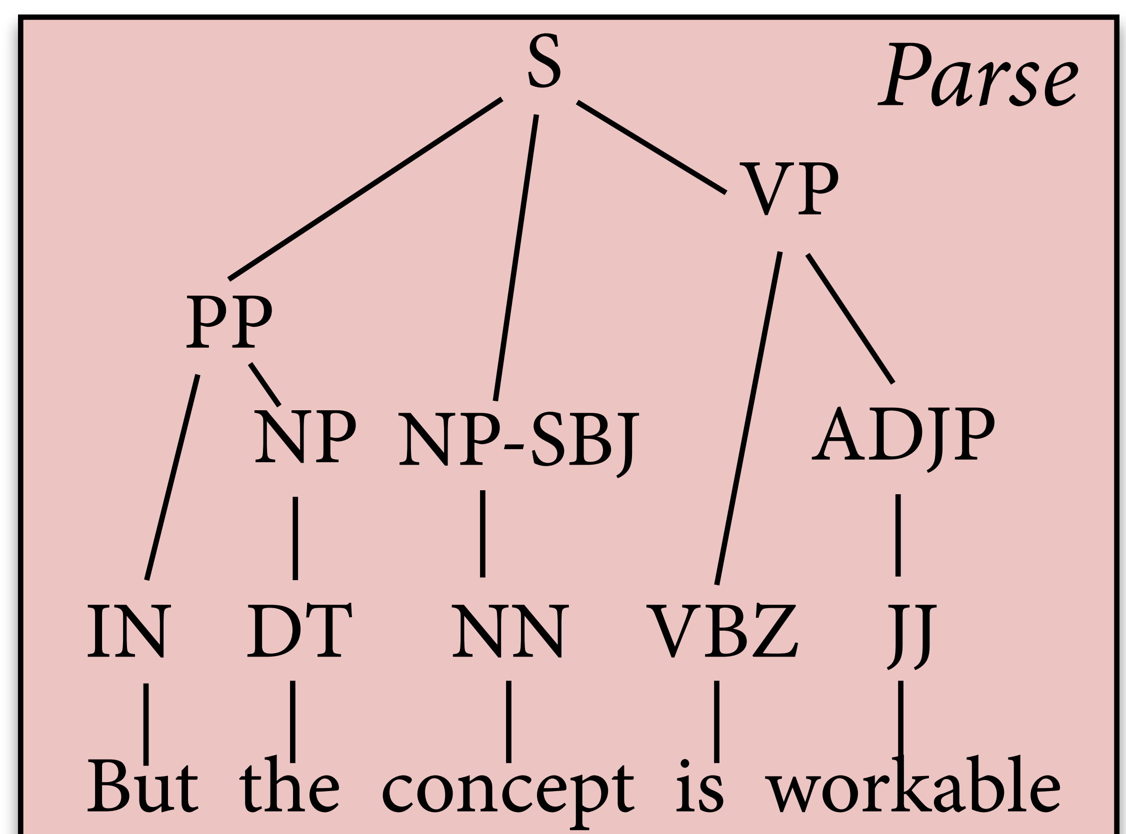
Evaluation

- Step 2: How should we measure the accuracy of the parser?
- Straightforward idea: Measure “exact match”, i.e. proportion of gold standard trees that parser got right.
- This is too strict:
  - parser makes many decisions in parsing a sentence
  - a single incorrect parsing decision makes tree “wrong”
  - want more fine-grained measure

Comparing parse trees

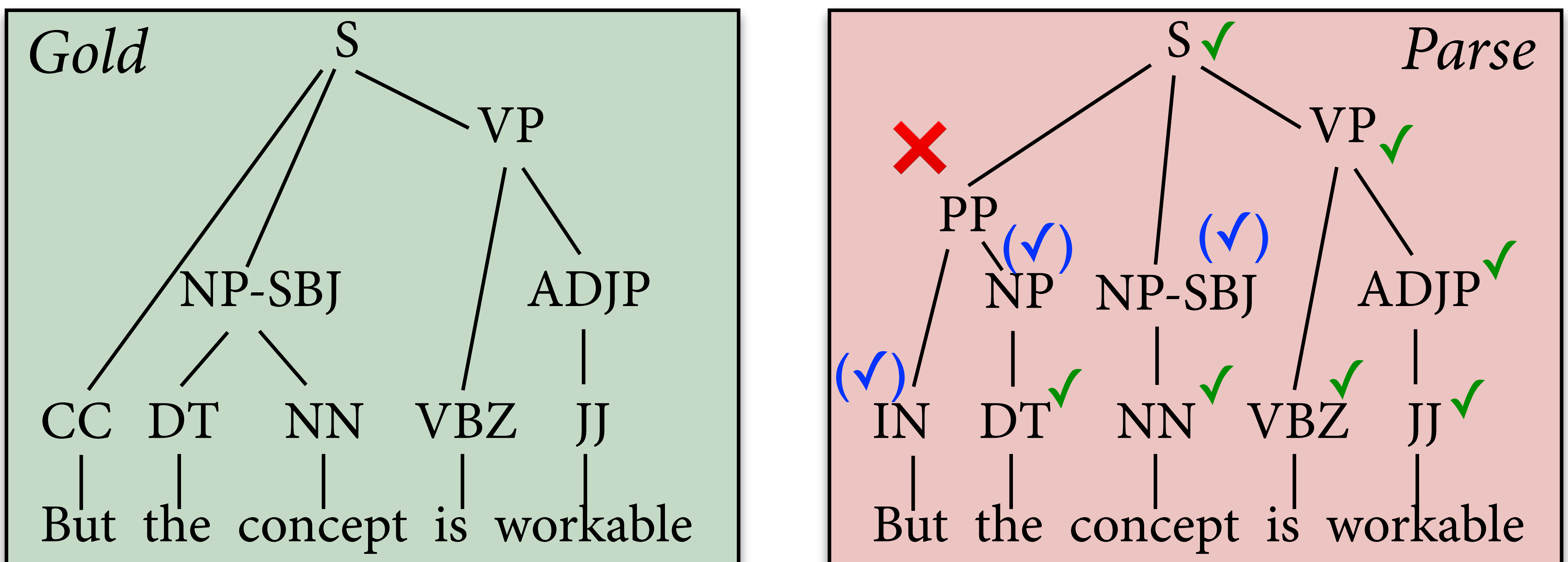
- Idea 2 (PARSEVAL): Compare *structure* of parse tree and gold standard tree.
  - Labeled: Which *constituents* (span + syntactic category) of one tree also occur in the other?
  - Unlabeled: How do the trees bracket the *substrings* of the sentence (ignoring syntactic categories)?





Precision

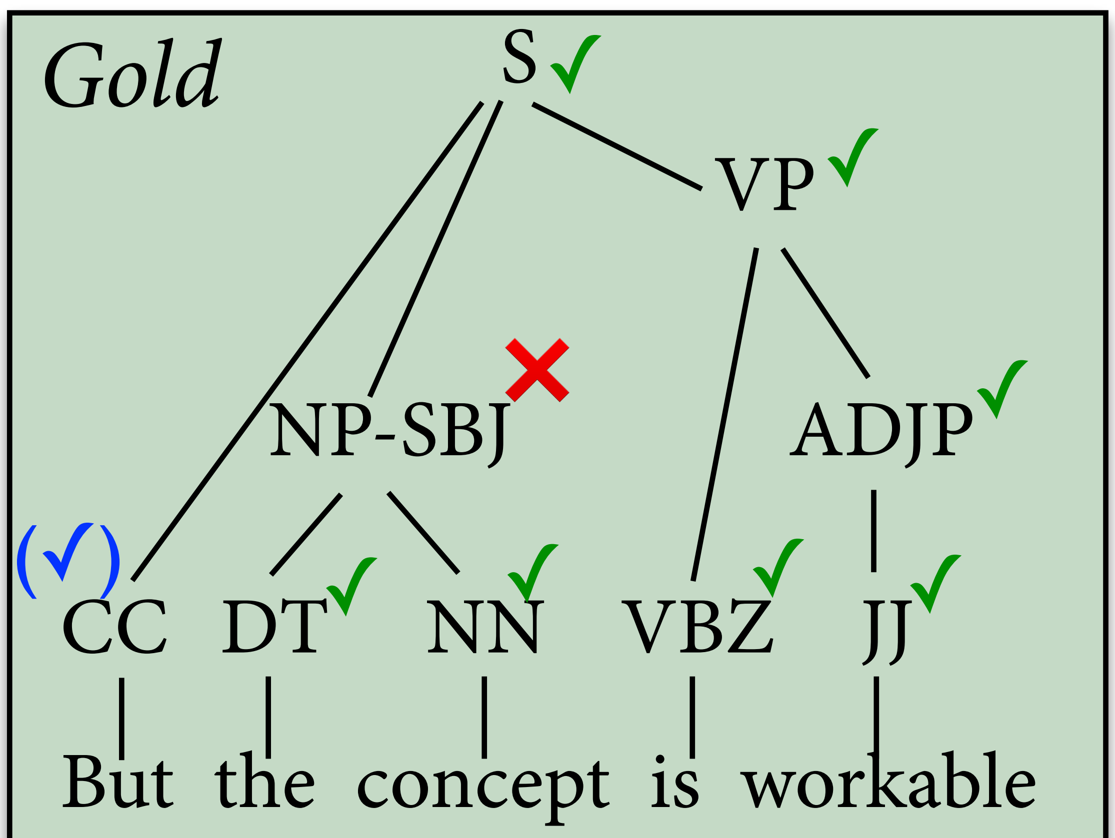
What proportion of constituents in *parse tree* is also present in *gold tree?*

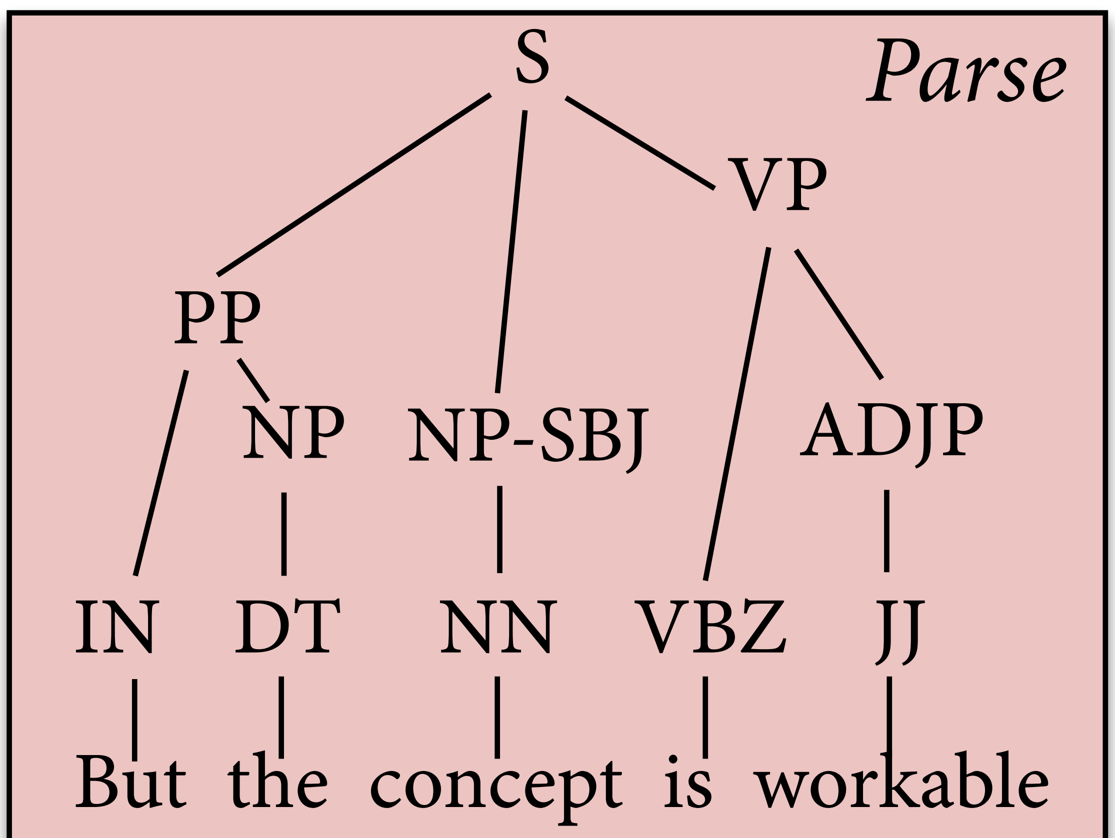


Labeled Precision = 7 / 11 = 63.6%
Unlabeled Precision = 10 / 11 = 90.9%

Recall

What proportion of constituents in *gold tree* is also present in *parse tree?*





Labeled Recall = 7 / 9 = 77.8%

Unlabeled Recall = 8 / 9 = 88.9%

F-Score

- Precision and recall measure opposing qualities of a parser ("soundness" and "completeness")

- Summarize both together in the *f-score:*

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

- In the example, we have labeled f-score 70.0 and unlabeled f-score 89.9.

Today

- Parameters of PCFG = rule probabilities.
- How do we learn parameters from corpora?
  - maximum likelihood estimation
  - “hard EM” using Viterbi
  - “soft EM” using the inside-outside algorithm

ML Estimation

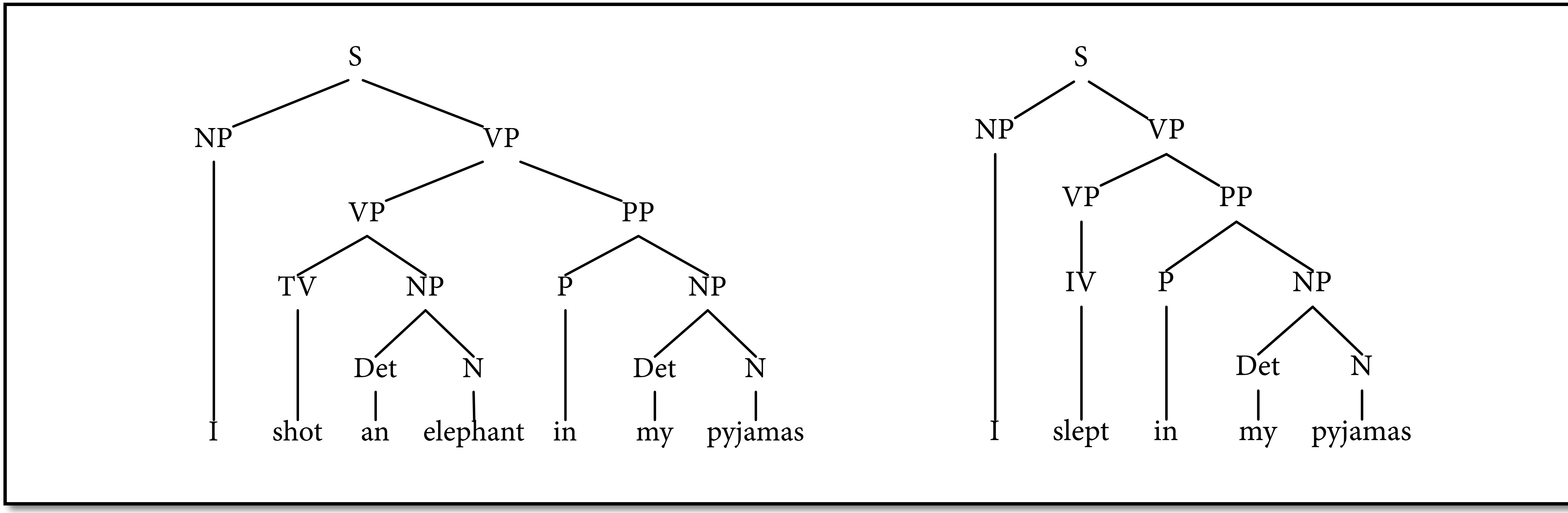
- Assume we have a treebank.
  - that is, every sentence annotated by hand with its "correct" parse tree

- Then we can use MLE to obtain rule probabilities:

$$P(A \to w) = \frac{C(A \to w)}{C(A \to \bullet)} = \frac{C(A \to w)}{\sum_{w'} C(A \to w')}$$

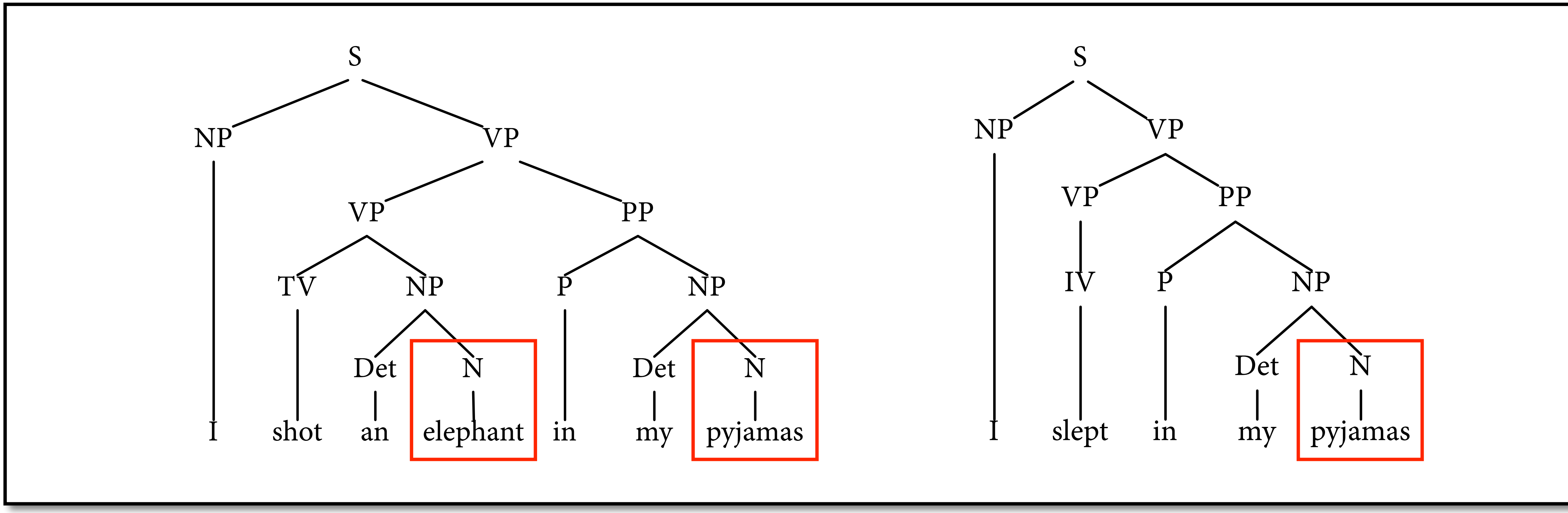
- Standard way of parameter estimation in practice. Works well, smoothing only needed for unknown words (or replace by POS tags).

Example



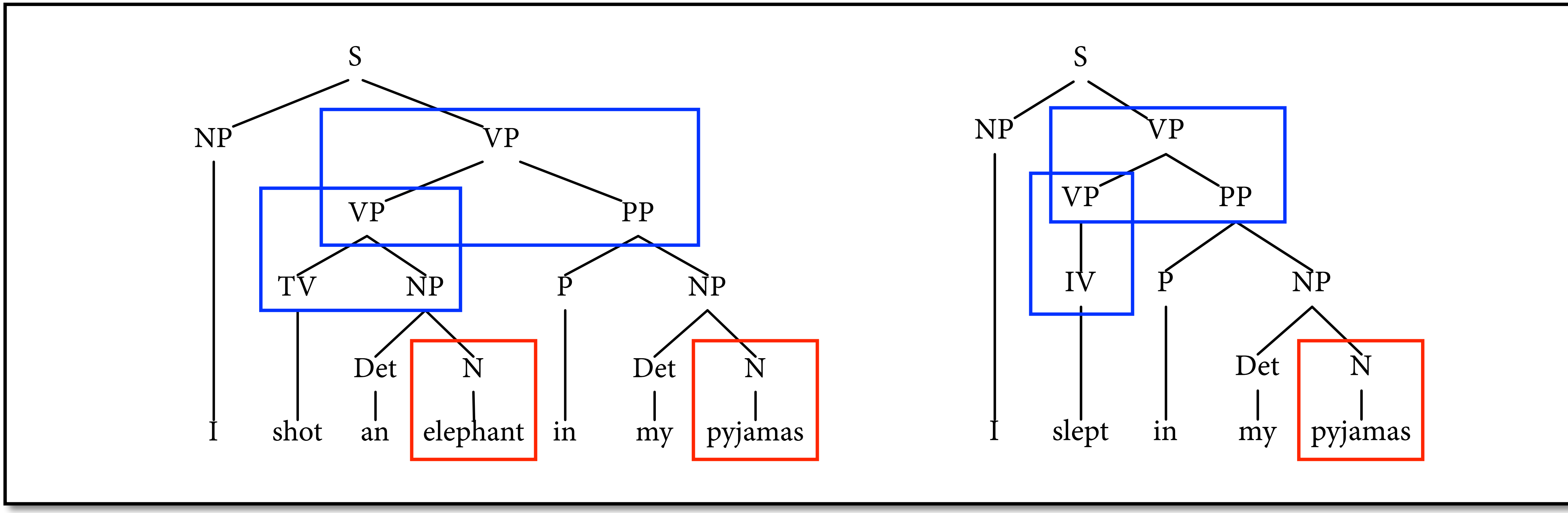
S
NP
VP
VP
PP
TV
NP
P
NP
Det
N
Det
N
I shot an elephant in my pyjamas
S
NP
VP
VP
PP
IV
P
NP
Det
N
I slept in my pyjamas

Example

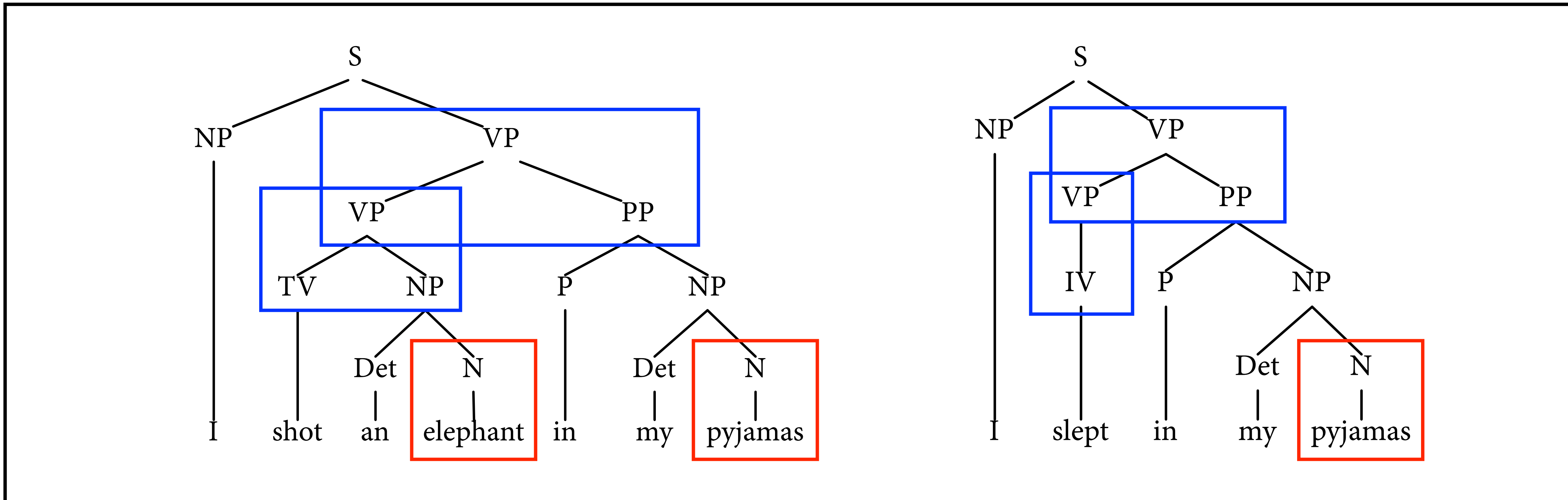


S
NP
VP
VP
PP
TV
NP
P
NP
Det
N
Det
N
I
shot
an
elephant
in
my
pyjamas
S
NP
VP
VP
PP
IV
P
NP
Det
N
I
slept
in
my
pyjamas

Example


S
NP
VP
VP
PP
TV
NP
P
NP
Det
N
Det
N
I
shot
an
elephant
in
my
pyjamas
S
NP
VP
VP
PP
IV
P
NP
Det
N
I
slept
in
my
pyjamas

Example



N → N PP	[0]	VP → TV NP	[1/4]
N → elephant	[1/3]	VP → IV	[1/4]
N → pyjamas	[2/3]	VP → VP PP	[1/2]

Unsupervised estimation

- MLE works okay *for English*.
  - German: Tiger treebank exists, but is hard for PCFGs, e.g. because of free word order.
  - most other languages: phrase structure annotations unavailable, expensive to create → unsupervised methods?

- Unsupervised methods:
  - provide CFG, learn parameters from unannotated corpus
  - show first "hard EM", then "soft EM"
  - ideas instructive and generalize to other problems

“Hard” aka Viterbi EM

- In the absence of syntactic annotations, learner must invent its own parse trees.

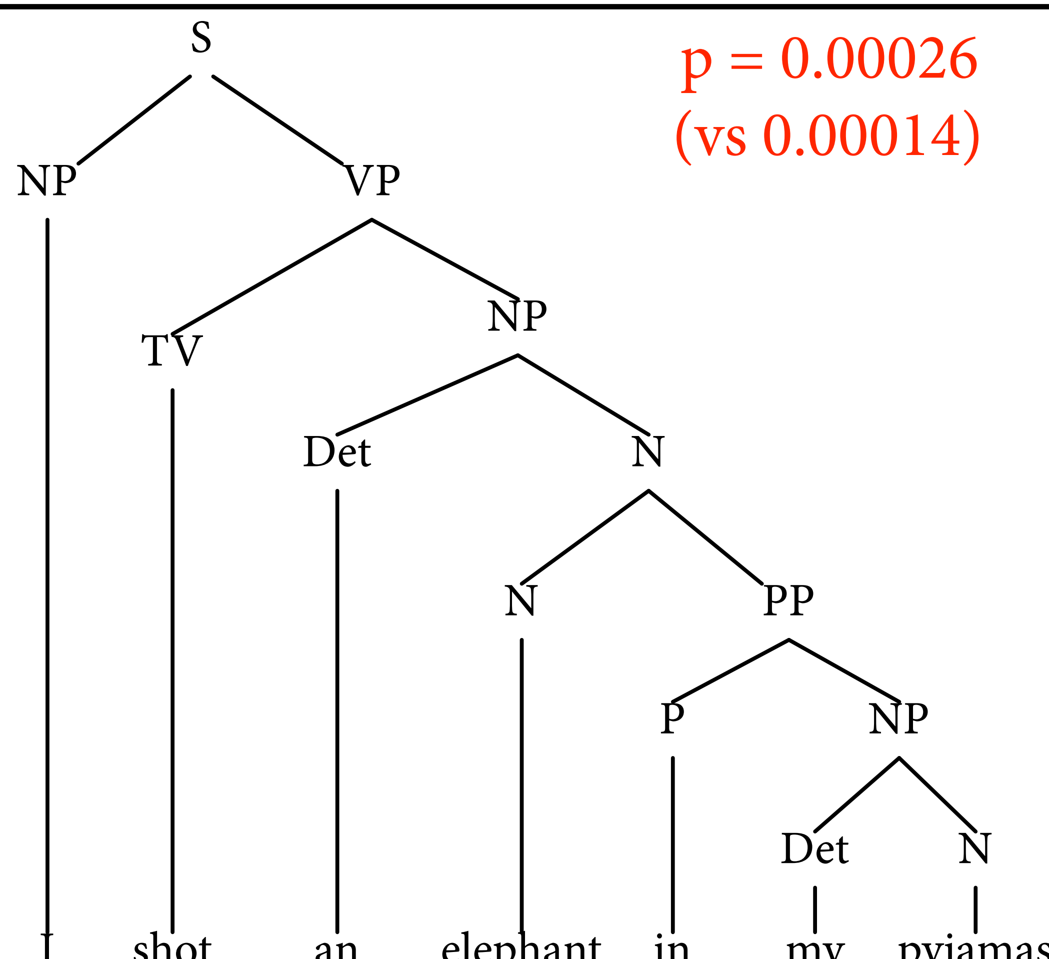
- Viterbi EM:
    - start with some parameter estimate
    - produce “syntactic annotations” by computing best tree for each sentence using Viterbi
    - apply MLE to re-estimate parameters
    - repeat as long as needed

- This is *not* real EM!

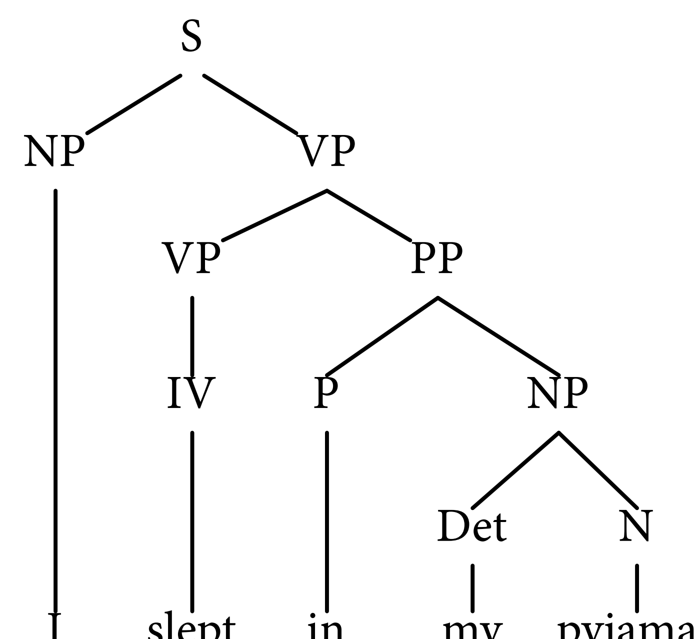
Example

1

N → N PP	[0.6]	VP → TV NP	[1/3]
N → elephant	[0.2]	VP → IV	[1/3]
N → pyjamas	[0.2]	VP → VP PP	[1/3]



p = 0.00026
(vs 0.00014)

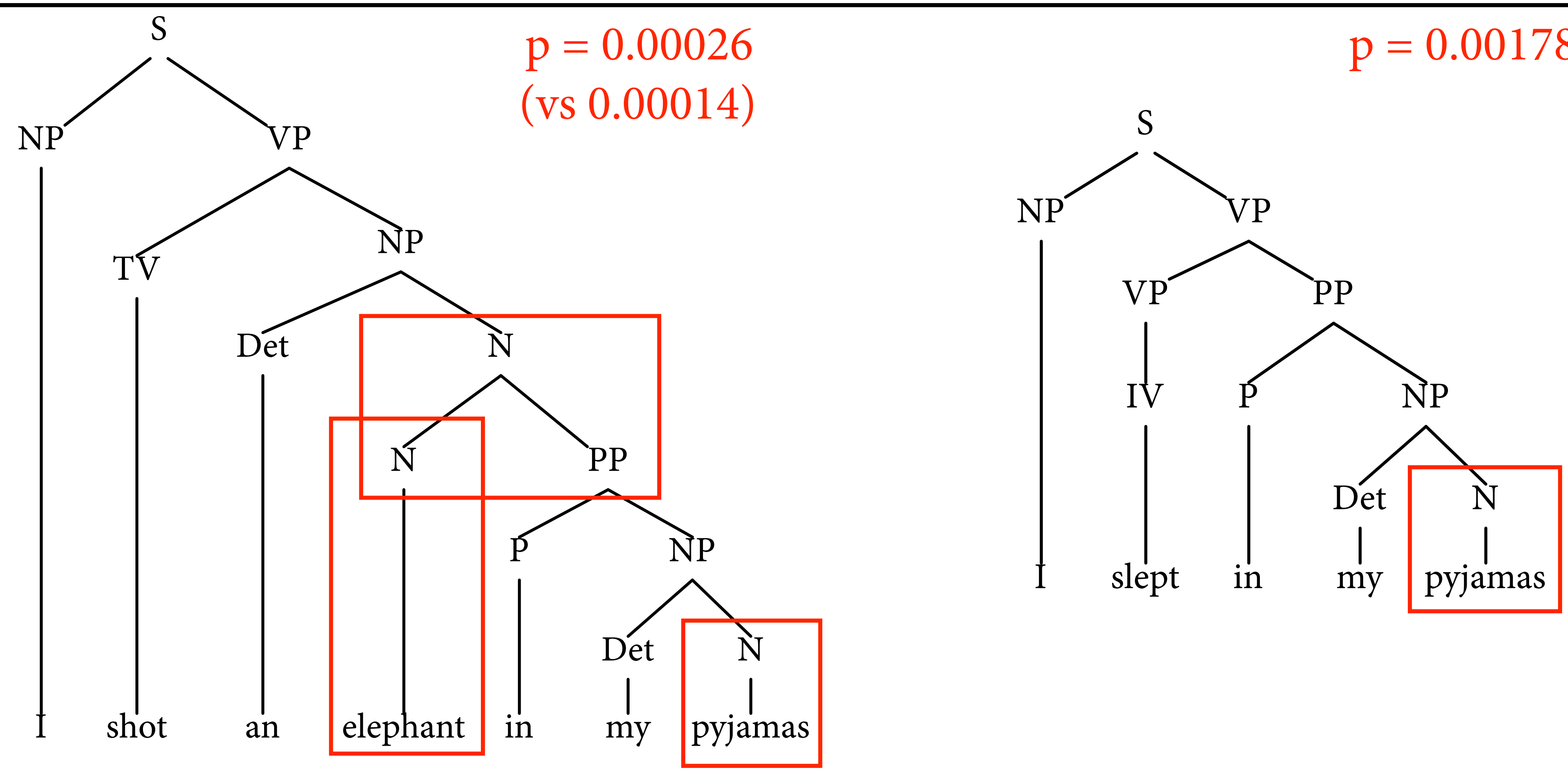


p = 0.00178

Example

1

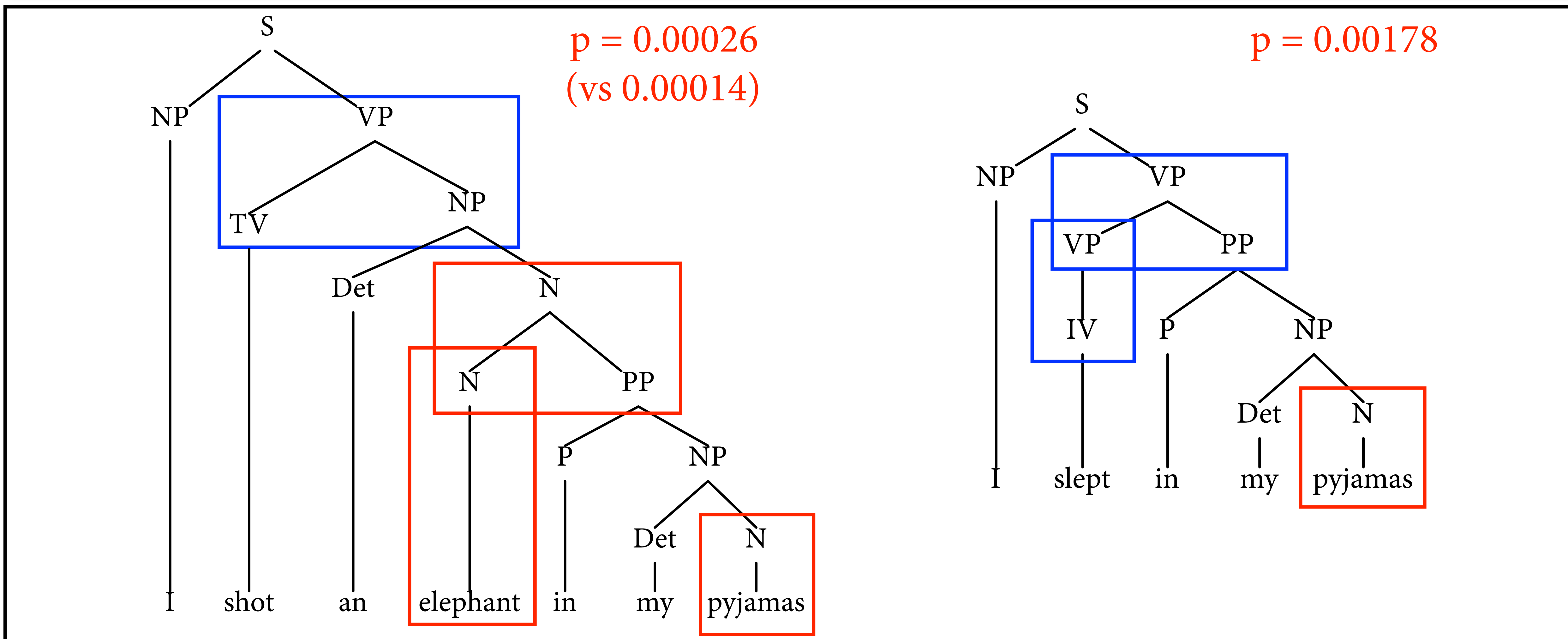
N → N PP	[0.6]	VP → TV NP	[1/3]
N → elephant	[0.2]	VP → IV	[1/3]
N → pyjamas	[0.2]	VP → VP PP	[1/3]



Example

1

N → N PP	[0.6]	VP → TV NP	[1/3]
N → elephant	[0.2]	VP → IV	[1/3]
N → pyjamas	[0.2]	VP → VP PP	[1/3]

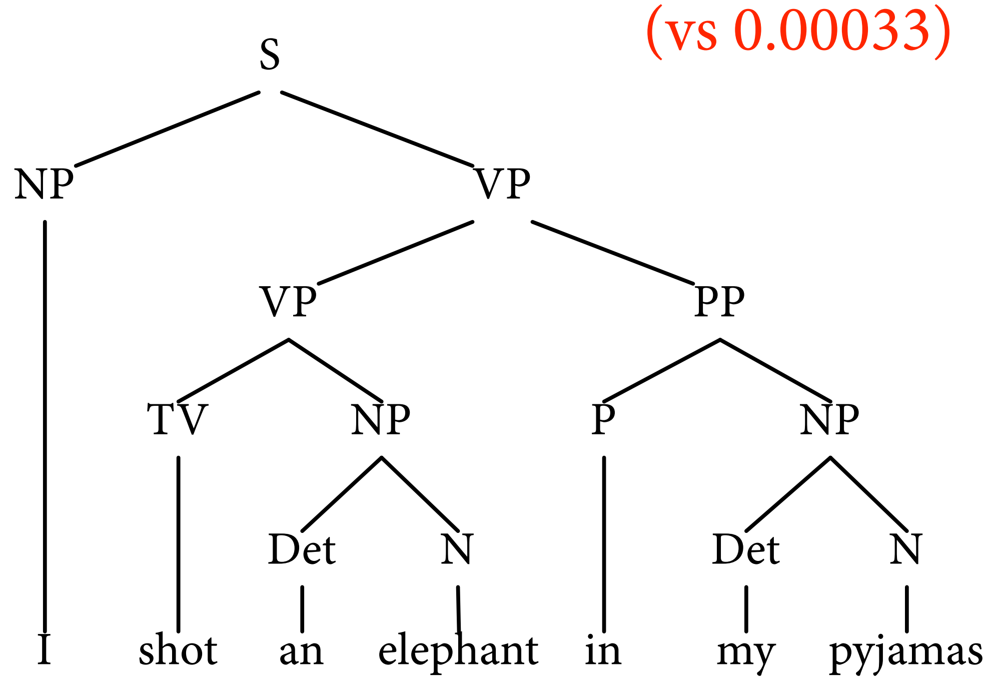


MLE on Viterbi parses

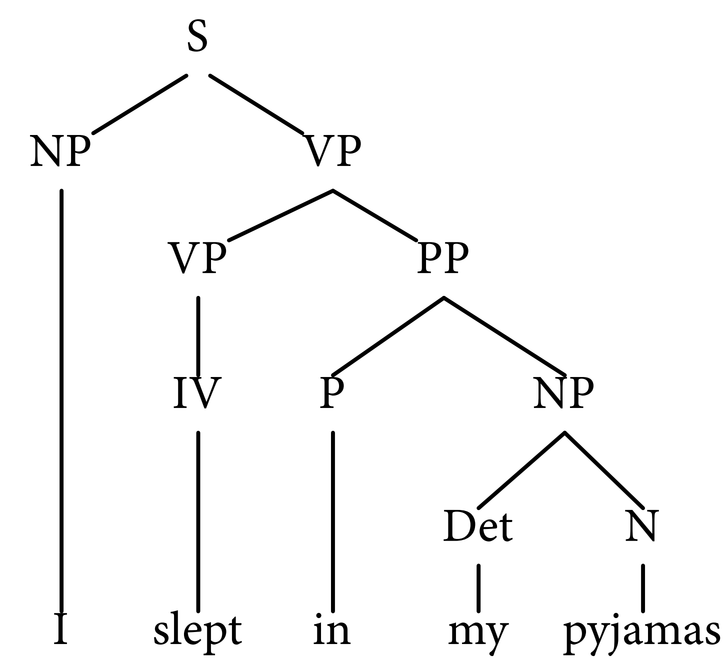
2

N → N PP	[1/4]	VP → TV NP	[1/3]
N → elephant	[1/4]	VP → IV	[1/3]
N → pyjamas	[1/2]	VP → VP PP	[1/3]

p = 0.00044
(vs 0.00033)



p = 0.00889



Some things to note

- In this example, the likelihood increased.
  - this need not always be the case for Viterbi EM
- Viterbi EM commits to a single parse tree per sentence. This has advantages and disadvantages:
  - parse tree easy to compute, and can simply apply MLE
  - ignores all uncertainty we had about correct parse (winning parse tree takes all)

Towards "real" (aka "soft")

idea: weighted counting of rules in all parse trees

Viterbi-EM

$$1 \cdot C_{t1}(r) \;\Big|\; + \; 0 \cdot C_{t2}(r) \; + \; 0 \cdot C_{t3}(r) \; + \; 0 \cdot C_{t4}(r)$$

EM

$$P(t_1 \mid w) \cdot C_{t1}(r) + P(t_2 \mid w) \cdot C_{t2}(r) + P(t_3 \mid w) \cdot C_{t3}(r) + P(t_4 \mid w) \cdot C_{t4}(r)$$

Expected counts

- Define *expected count* of rule A → B C, based on previous parameter estimate.

$$E(A \to B\ C) = \sum_{t \in \mathcal{T}} P(t \mid w) \cdot C_t(A \to B\ C)$$

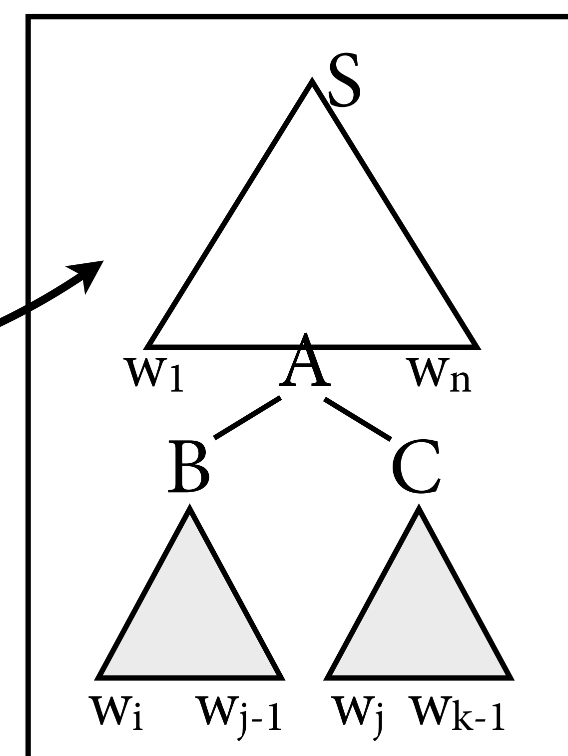
- If we have them, can re-estimate parameters:

$$P(A \to B\ C) = \frac{E(A \to B\ C)}{\sum_r E(A \to r)}$$

- Challenge: How to compute E(A → B C) efficiently?
  - we assume grammars in CNF here

Fundamental idea

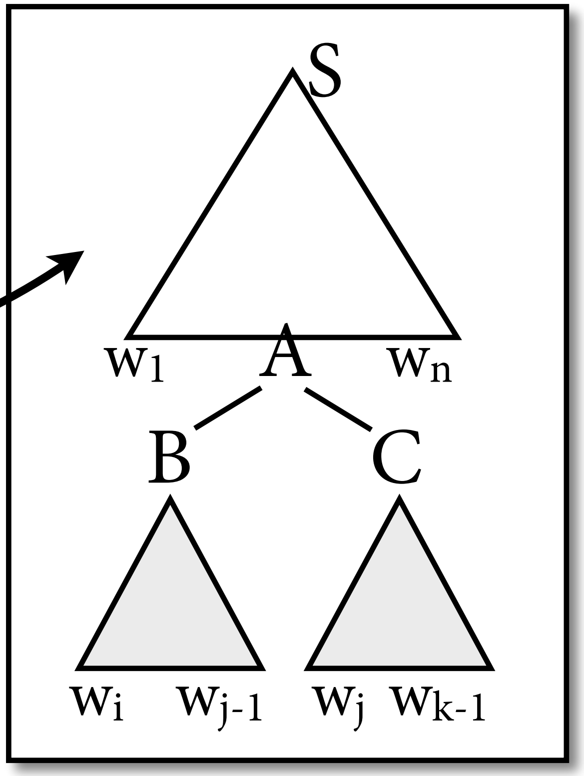
$$
\begin{aligned}
E(A \to B\ C) &= \sum_{t \in \mathcal{T}} P(t \mid w) \cdot C_t(A \to B\ C) \\
&= \frac{1}{P(w)} \sum_{t \in \mathcal{T}} P(t) \cdot C_t(A \to B\ C) \\
&= \frac{1}{P(w)} \sum_{t \in \mathcal{T}} P(t) \cdot \sum_{i,j,k} ||\text{rule for } i, j, k \text{ in } t \text{ is } A \to B\ C|| \\
&= \frac{1}{P(w)} \sum_{i,j,k} \left(\sum_{t \in \mathcal{T}} P(t) \cdot ||\text{rule for } i, j, k \text{ in } t \text{ is } A \to B\ C|| \right) \\
&= \frac{1}{P(w)} \sum_{i,j,k} \left(\sum_{t \text{ of this form}} P(t) \right)
\end{aligned}
$$



(note that P(t, w) = P(t))

Fundamental idea

$$E(A \to B\ C) = \sum_{t \in \mathcal{T}} P(t \mid w) \cdot C_t(A \to B\ C)$$
$$= \frac{1}{P(w)} \sum_{t \in \mathcal{T}} P(t) \cdot C_t(A \to B\ C)$$
$$= \frac{1}{P(w)} \sum_{t \in \mathcal{T}} P(t) \cdot \sum_{i,j,k} ||\text{rule for } i,j,k \text{ in } t \text{ is } A \to B\ C||$$
$$= \frac{1}{P(w)} \sum_{i,j,k} \left(\sum_{t \in \mathcal{T}} P(t) \cdot ||\text{rule for } i,j,k \text{ in } t \text{ is } A \to B\ C|| \right)$$
$$= \frac{1}{P(w)} \sum_{i,j,k} \left(\sum_{t \text{ of this form}} P(t) \right)$$

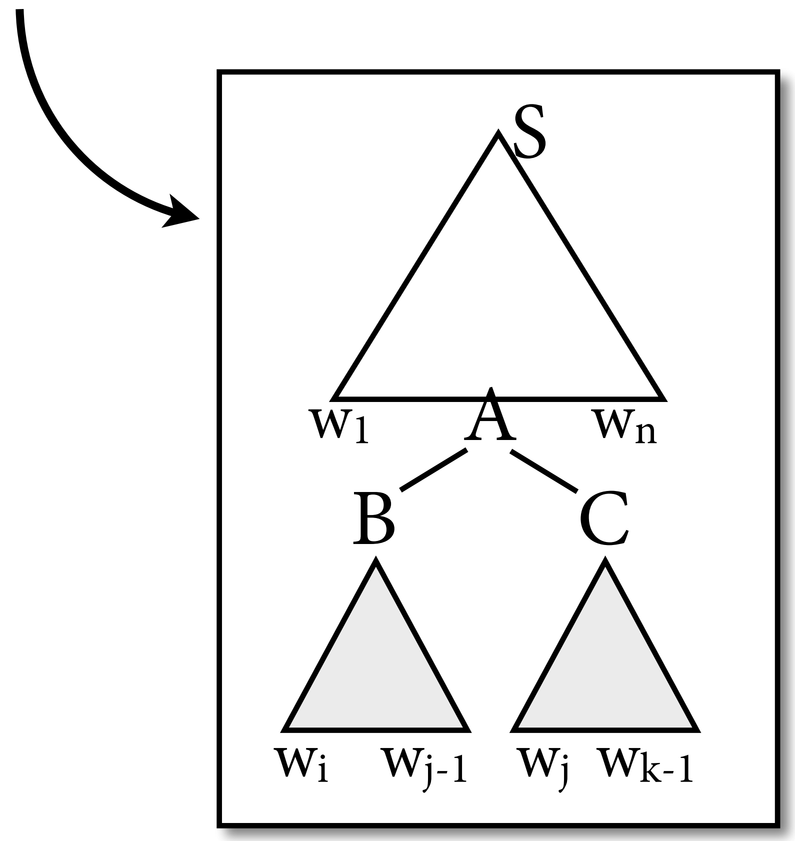


call this term $\mu(A \to B\ C, i, j, k)$

(note that $P(t, w) = P(t)$)

Computing μ

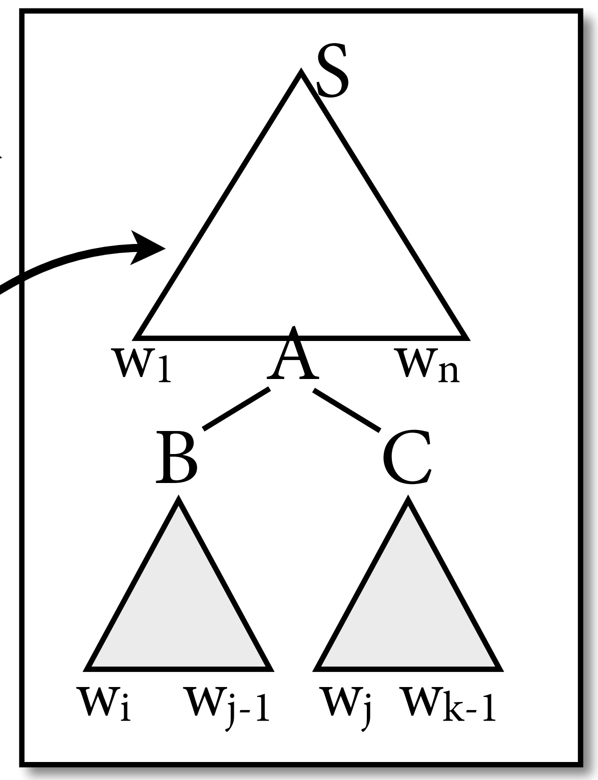
$$\mu(A \to B\ C, i, j, k) = \sum_{t \text{ of this form}} P(t)$$



Computing μ

$$\mu(A \to B\ C, i, j, k) = \sum_{t \text{ of this form}} P(t)$$

d_1: $S \Rightarrow^* w_1 \ldots w_{i-1}\ A\ w_k \ldots w_n$

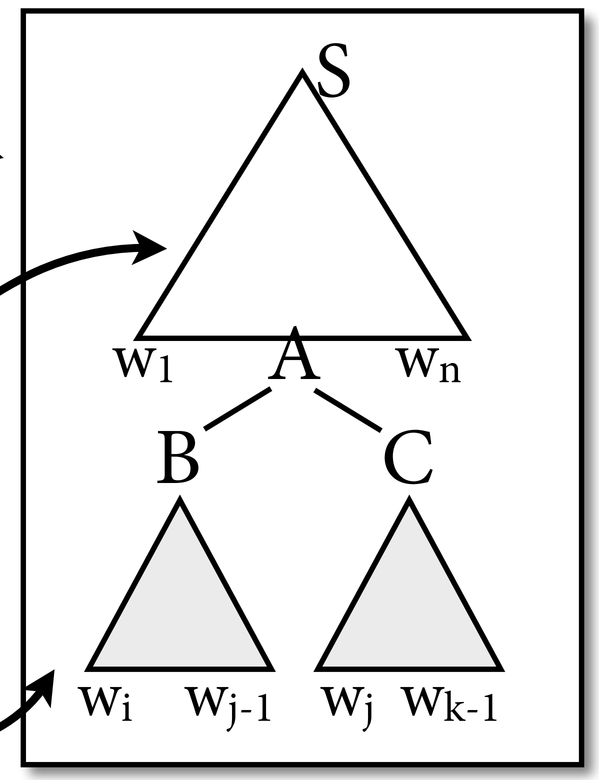


Computing μ

$$\mu(A \to B\ C, i, j, k) = \sum_{t \text{ of this form}} P(t)$$

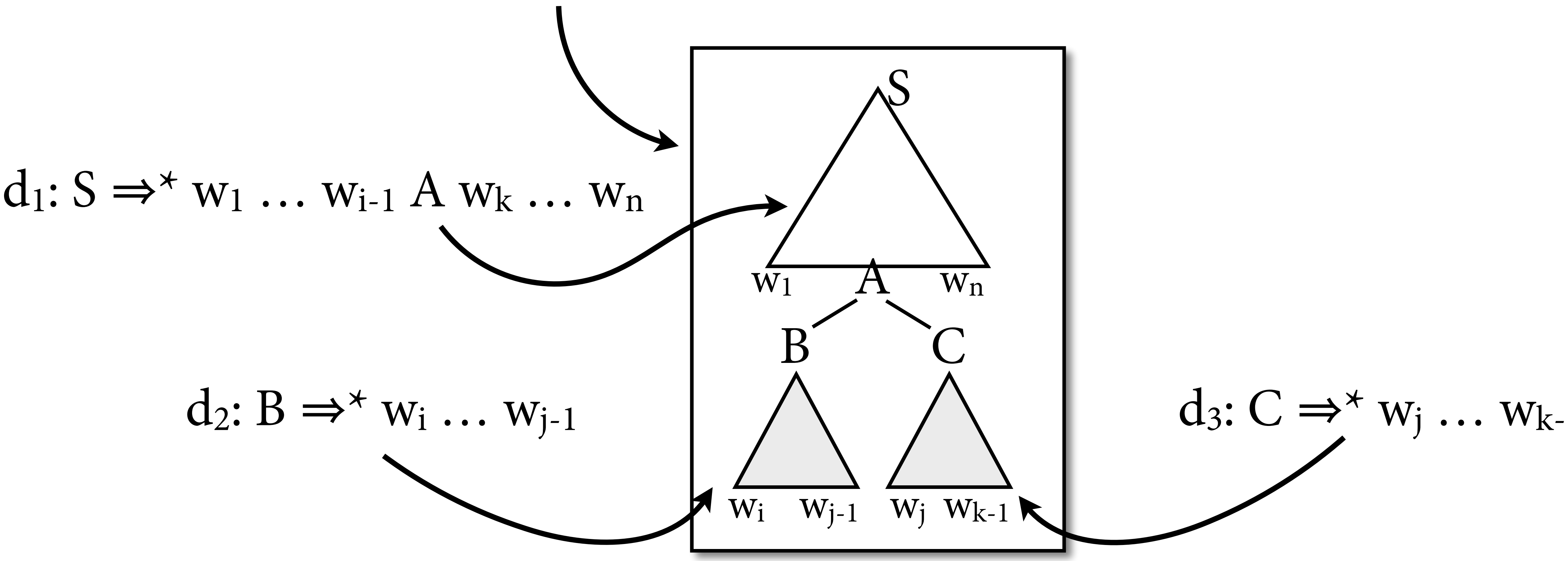
d_1: S ⇒* w_1 … w_{i-1} A w_k … w_n

d_2: B ⇒* w_i … w_{j-1}



Computing μ

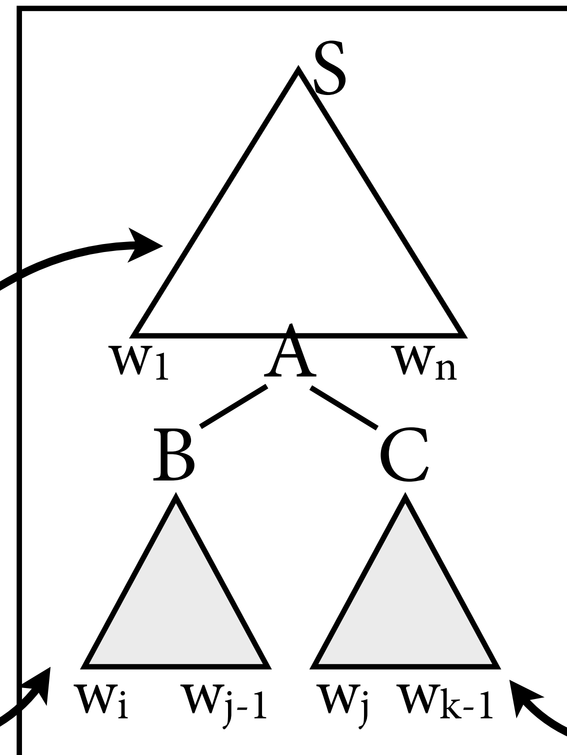
$$\mu(A \to B\ C, i, j, k) = \sum_{t \text{ of this form}} P(t)$$



Computing μ

$$\mu(A \to B\ C, i, j, k) = \sum_{t \text{ of this form}} P(t)$$

d_1: S ⇒* w_1 … w_{i-1} A w_k … w_n



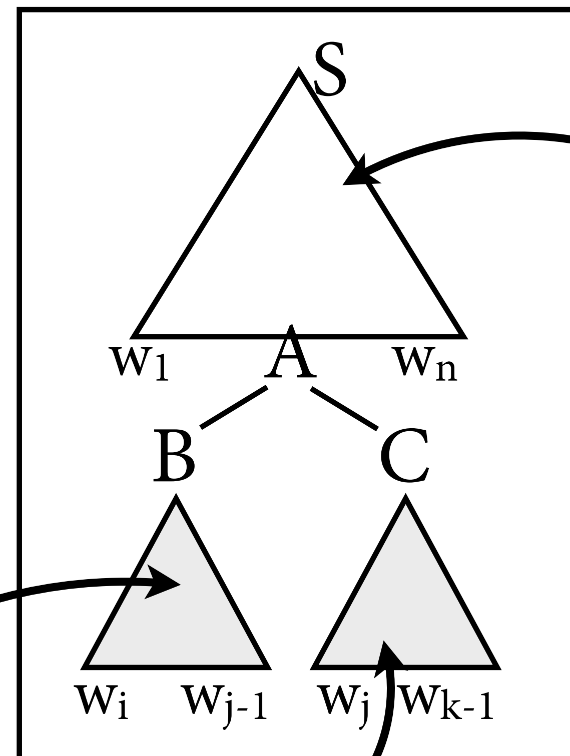
d_2: B ⇒* w_i … w_{j-1}

d_3: C ⇒* w_j … w_{k-1}

$$\begin{aligned}
\mu(A \to B\ C, i, j, k) &= \sum_{t \text{ of this form}} P(t) \\
&= \sum_{t \text{ of this form}} P(d_1) \cdot P(A \to B\ C) \cdot P(d_2) \cdot P(d_3) \\
&= \left(\sum_{d_1} P(d_1)\right) \cdot P(A \to B\ C) \cdot \left(\sum_{d_2} P(d_2)\right) \cdot \left(\sum_{d_3} P(d_3)\right)
\end{aligned}$$

Computing μ

$$\mu(A \to B\ C, i, j, k) = \sum_{t \text{ of this form}} P(t) = \alpha(A, i, k) \cdot P(A \to B\ C) \cdot \beta(B, i, j) \cdot \beta(C, j, k)$$



inside probability

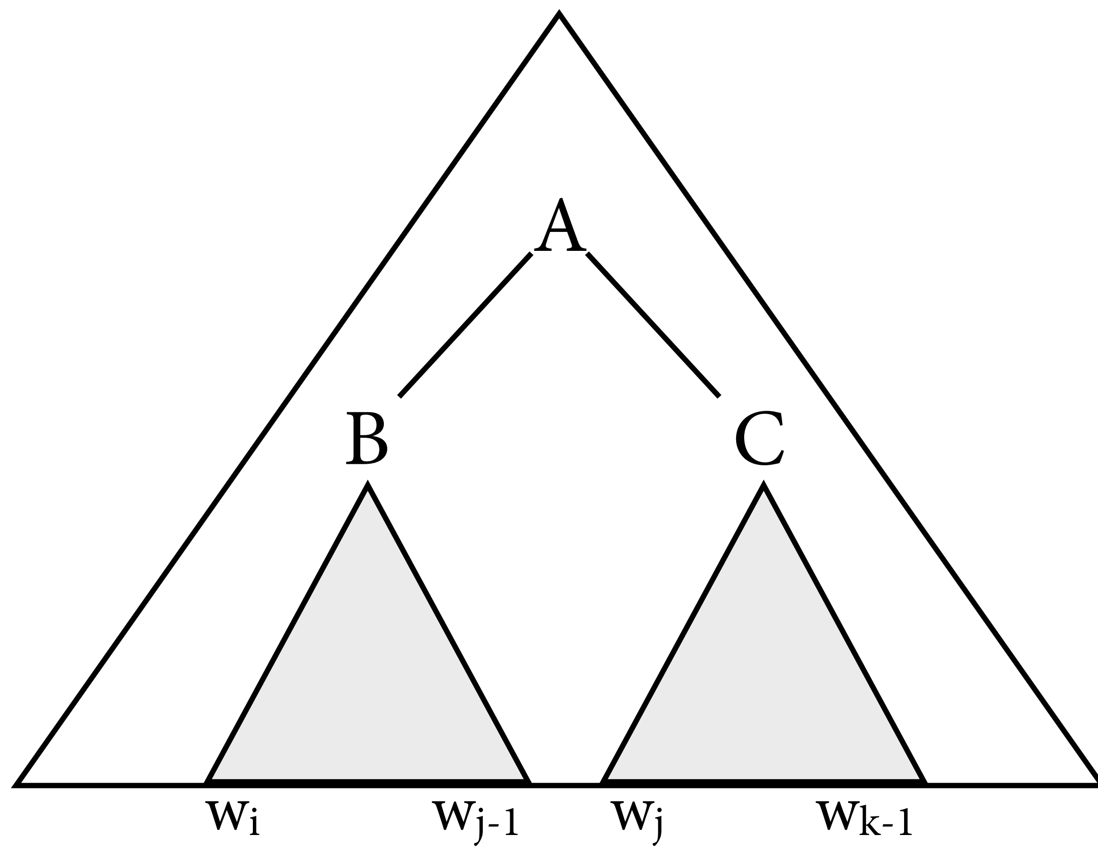
$$\beta(B, i, j) = \sum_{B \overset{d}{\Rightarrow}^{*} w_i \ldots w_{j-1}} P(d)$$

outside probability

$$\alpha(A, i, k) = \sum_{S \overset{d}{\Rightarrow}^{*} w_1 \ldots w_{i-1} A w_k \ldots w_n} P(d)$$

Inside probabilities

$$\beta(B,i,j) = \sum_{B \overset{d}{\Rightarrow}^* w_i \dots w_{j-1}} P(d)$$



special case:
P(w) = β(S,1,n+1)

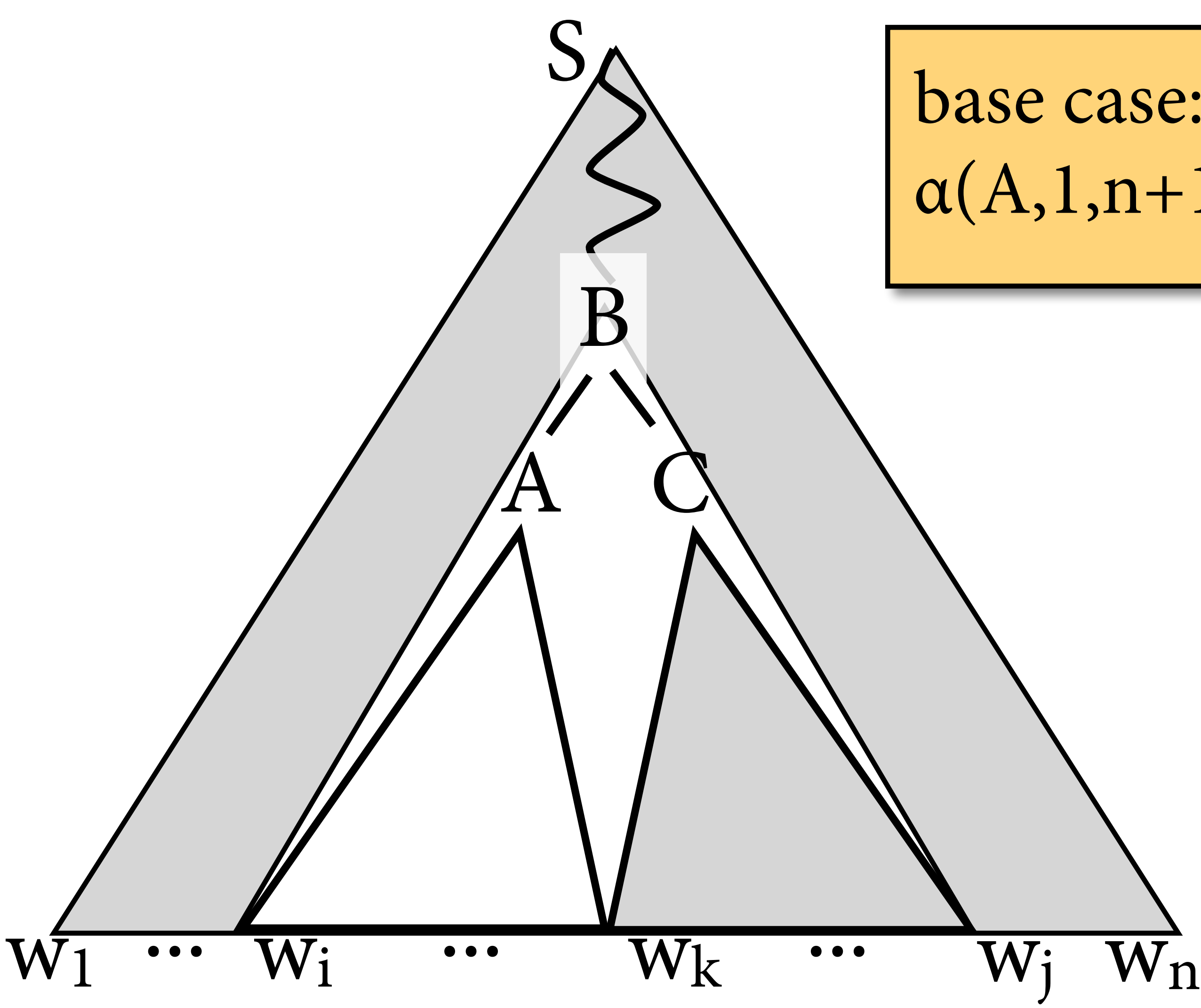
$$\beta(A,i,i+1) = P(A \rightarrow w_i)$$

$$\beta(A,i,k) = \sum_{\substack{A \rightarrow B\ C \\ i<j<k}} P(A \rightarrow B\ C) \cdot \beta(B,i,j) \cdot \beta(C,j,k)$$

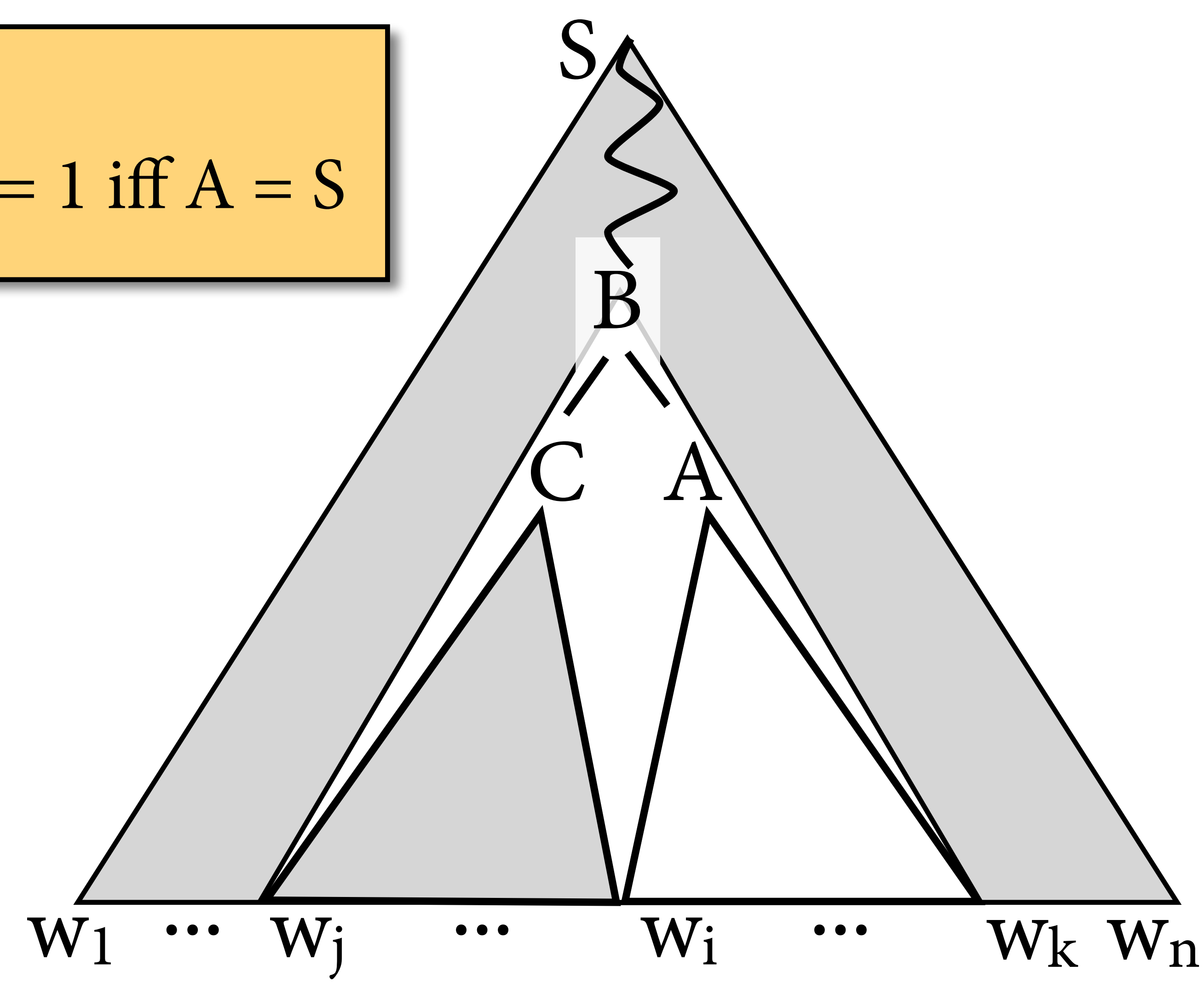
Outside probabilities

$$\alpha(A,i,k) = \sum_{S \overset{d}{\Rightarrow}^{*} w_1 \ldots w_{i-1} A w_k \ldots w_n} P(d)$$

$$= \sum_{\substack{B \to A\ C \\ k < j \leq n}} P(B \to A\ C) \cdot \beta(C,k,j) \cdot \alpha(B,i,j) + \sum_{\substack{B \to C\ A \\ 1 \leq j < i}} P(B \to C\ A) \cdot \beta(C,j,i) \cdot \alpha(B,j,k)$$



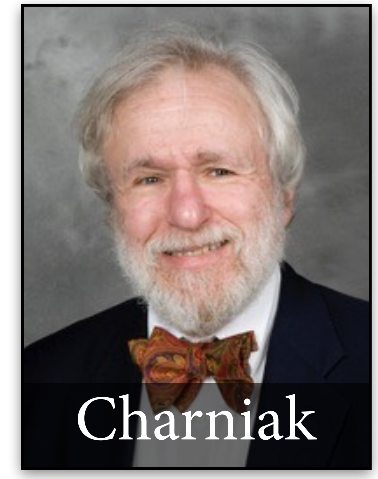
base case:
$\alpha(A,1,n+1) = 1$ iff $A = S$



The Inside-Outside Algorithm

- Start with some initial estimate of parameters.
- For each sentence w, compute α, β, and μ.
- Compute expected counts E(A → B C).
    - sum expected counts over all sentences
    - remember that P(w) = β(S, 1, n+1)
- Re-estimate P(A → B C) from expected counts.
- Iterate until convergence.

Some remarks



- Inside-outside increases likelihood in each step.
- But huge problems with local maxima.
  - Carroll & Charniak 92 find 300 different local maxima for 300 different initial parameter estimates.
  - Improve by partially bracketing strings (Pereira & Schabes 92).
- Therefore, EM doesn't really work for totally unsupervised PCFG training.
- But extremely useful in refining existing grammars (Berkeley parser; see next time).

Summary

- Learning parameters of PCFGs:
  - maximum likelihood estimation from raw text
  - "hard EM": iterate MLE on Viterbi parses
  - EM: use inside-outside algorithm with expected rule counts
- PCFG parsing with MLE parse gets f-score in low 70's. Will improve on this next time (state of the art: 93).
- Have assumed that CFG is given and only parameters are to be learned. Will fix this later in this course.