CS447: Natural Language Processing
http://courses.engr.illinois.edu/cs447

Lecture 10: Statistical Parsing with PCFGs

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center

Where we're at

Previous lecture:
Standard **CKY** (for non-probabilistic CFGs)
The standard CKY algorithm finds all possible parse trees τ for a sentence $S = w^{(1)} \ldots w^{(n)}$ under a CFG G in Chomsky Normal Form.

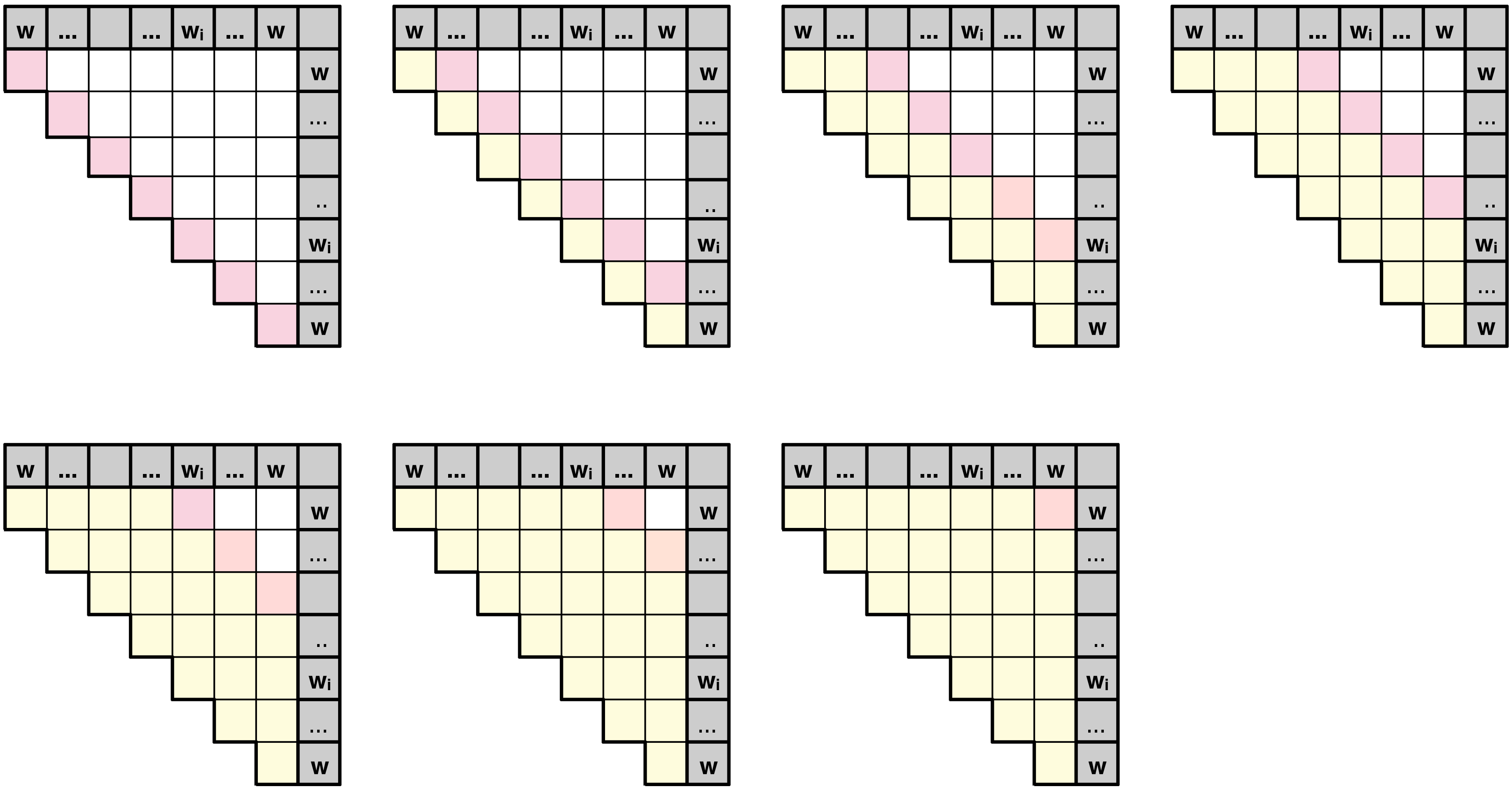
Today's lecture:

Probabilistic Context-Free Grammars (PCFGs)
– CFGs in which each rule is associated with a probability

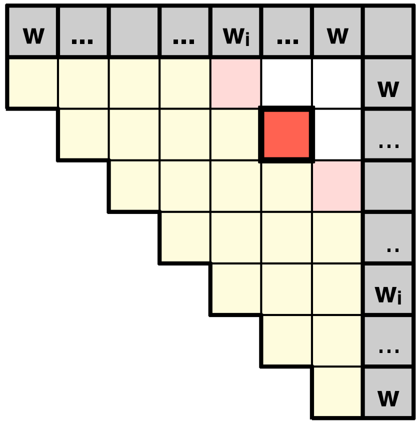
CKY for PCFGs (Viterbi):
– CKY for PCFGs finds the most likely parse tree $\tau^* = \text{argmax } P(\tau \mid S)$ for the sentence S under a PCFG.

Previous Lecture: CKY for CFGs

CKY: filling the chart



CKY: filling one cell

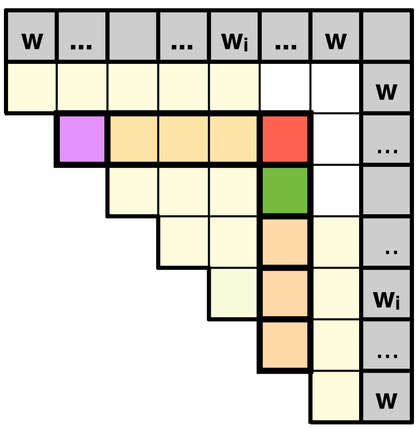


chart[2][6]:

w_1 $\mathbf{w_2}$ $\mathbf{w_3}$ $\mathbf{w_4}$ $\mathbf{w_5}$ $\mathbf{w_6}$ w_7

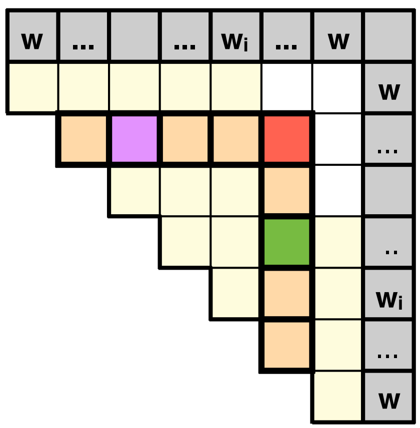
chart[2][6]:

w_1 $\mathbf{w_2 w_3 w_4 w_5 w_6}$ w_7



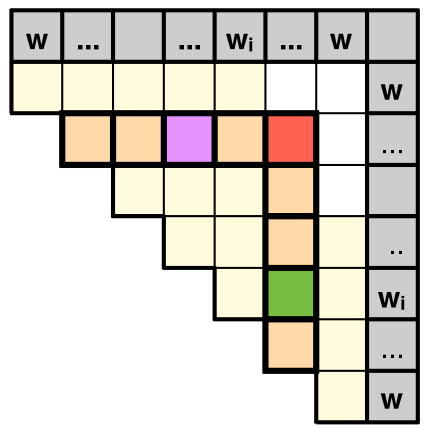
chart[2][6]:

w_1 $\mathbf{w_2 w_3 w_4 w_5 w_6}$ w_7



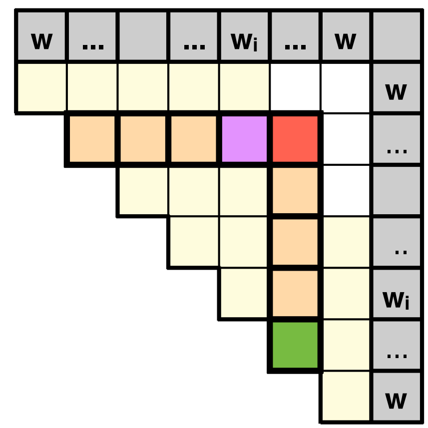
chart[2][6]:

w_1 $\mathbf{w_2 w_3 w_4 w_5 w_6}$ w_7



chart[2][6]:

w_1 $\mathbf{w_2 w_3 w_4 w_5 w_6}$ w_7



CKY for standard CFGs

CKY is a bottom-up chart parsing algorithm that finds all possible parse trees τ for a sentence $S = w^{(1)} \ldots w^{(n)}$ under a CFG *G* in Chomsky Normal Form (CNF).

- **CNF**: *G* has two types of rules: $X \longrightarrow Y\ Z$ and $X \longrightarrow w$ (X, Y, Z are nonterminals, w is a terminal)
- CKY is a **dynamic programming** algorithm
- The **parse chart** is an n×n upper triangular matrix: Each cell chart[i][j] ($i \leq j$) stores **all subtrees** for $w^{(i)} \ldots w^{(j)}$
- Each cell chart[i][j] has at most **one entry for each nonterminal** X (and **pairs of backpointers** to each pair of (Y, Z) entry in cells chart[i][k] chart[k+1][j] from which an X can be formed
- Time Complexity: $O(n^3 |G|)$

Dealing with ambiguity: Probabilistic Context-Free Grammars (PCFGs)

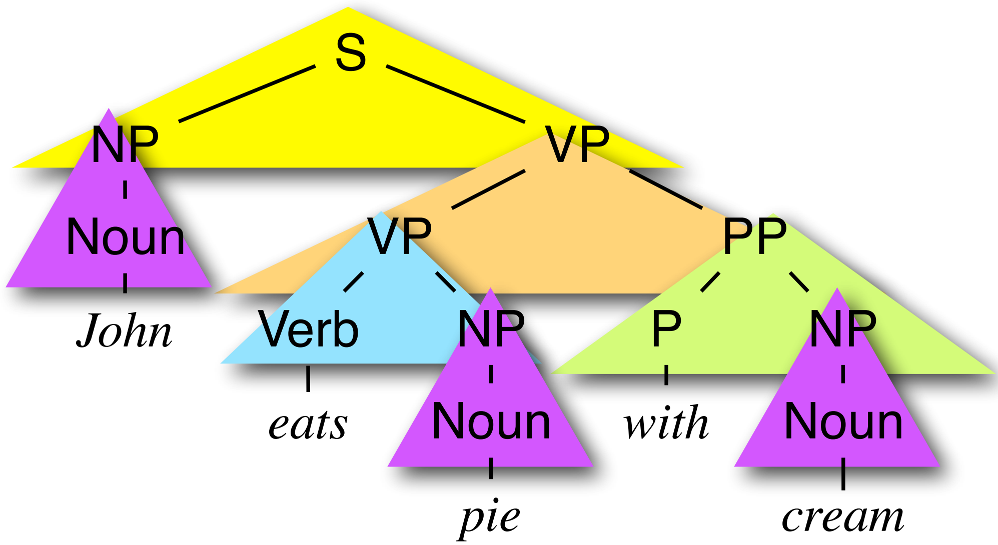
Probabilistic Context-Free Grammars

For every nonterminal X, define a probability distribution $P(X \rightarrow \alpha \mid X)$ over all rules with the same LHS symbol X:

S	$\longrightarrow$	NP VP	0.8
S	$\longrightarrow$	S conj S	0.2
NP	$\longrightarrow$	Noun	0.2
NP	$\longrightarrow$	Det Noun	0.4
NP	$\longrightarrow$	NP PP	0.2
NP	$\longrightarrow$	NP conj NP	0.2
VP	$\longrightarrow$	Verb	0.4
VP	$\longrightarrow$	Verb NP	0.3
VP	$\longrightarrow$	Verb NP NP	0.1
VP	$\longrightarrow$	VP PP	0.2
PP	$\longrightarrow$	P NP	1.0

Computing $P(\tau)$ with a PCFG

The probability of a tree τ is the product of the probabilities of all its rules:



S	$\longrightarrow$	NP VP	0.8
S	$\longrightarrow$	S conj S	0.2
NP	$\longrightarrow$	Noun	0.2
NP	$\longrightarrow$	Det Noun	0.4
NP	$\longrightarrow$	NP PP	0.2
NP	$\longrightarrow$	NP conj NP	0.2
VP	$\longrightarrow$	Verb	0.4
VP	$\longrightarrow$	Verb NP	0.3
VP	$\longrightarrow$	Verb NP NP	0.1
VP	$\longrightarrow$	VP PP	0.2
PP	$\longrightarrow$	P NP	1.0

$$P(\tau) = 0.8 \times 0.3 \times 0.2 \times 1.0 \times 0.2^3$$

$$= \mathbf{0.00384}$$

Learning the parameters of a PCFG

If we have a treebank (a corpus in which each sentence is associated with a parse tree), we can just count the number of times each rule appears, e.g.:

```
S → NP VP .     (count = 1000)
S → S conj S .  (count = 220)
```

and then we divide the observed frequency of each rule X → Y Z by the sum of the frequencies of all rules with the same LHS X to turn these counts into probabilities:

```
S → NP VP .     (p = 1000/1220)
S → S conj S .  (p = 220/1220)
```

More on probabilities:

Computing $P(s)$:

If $P(\tau)$ is the probability of a tree τ,
the probability of a sentence s is the sum of the probabilities of all its parse trees:

$$P(s) = \sum_{\tau:yield(\tau) = s} P(\tau)$$

How do we know that $P(L) = \sum_{\tau} P(\tau) = 1$?

If we have learned the PCFG from a corpus via MLE, this is guaranteed to be the case.

If we just set the probabilities by hand, we could run into trouble, as in the following example:

```
S → S S  (0.9)
S → w (0.1)
```

PCFG parsing (decoding): Probabilistic CKY

Probabilistic CKY: Viterbi

Like standard CKY, but with probabilities.
Finding the most likely tree is similar to Viterbi for HMMs:

Initialization:

- [*optional*] Every chart entry that corresponds to a **terminal** (entry w in `cell[i][i]`) has a Viterbi probability $P_{\text{VIT}}(w_{[i][i]}) = 1$ (*)
- Every entry for a **non-terminal** `X` in `cell[i][i]` has Viterbi probability $P_{\text{VIT}}(X_{[i][i]}) = P(X \rightarrow w \mid X)$ [and a single backpointer to $w_{[i][i]}$ (*)]

Recurrence: For every entry that corresponds to a **non-terminal** `X` in `cell[i][j]`, keep only the highest-scoring pair of backpointers to any pair of children (`Y` in `cell[i][k]` and `Z` in `cell[k+1][j]`):
$P_{\text{VIT}}(X_{[i][j]}) = \text{argmax}_{Y,Z,k}\, P_{\text{VIT}}(Y_{[i][k]}) \times P_{\text{VIT}}(Z_{[k+1][j]}) \times P(X \rightarrow Y\, Z \mid X)$

Final step: Return the Viterbi parse for the start symbol S in the top `cell[1][n]`.

*this is unnecessary for simple PCFGs, but can be helpful for more complex probability models

Probabilistic CKY

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
Noun 1.0 NP 0.2	S 0.8 · 0.2 · 0.3	S 0.8 · 0.2 · 0.06		S 0.2 · 0.0036 · 0.8	**John**
	Verb 1.0 VP 0.3	VP 1 · 0.3 · 0.2 = 0.06		VP max(1.0 · 0.008 · 0.3, 0.06 · 0.2 · 0.3)	**eats**
		Noun 1.0 NP 0.2		NP 0.2 · 0.2 · 0.2 = 0.008	**pie**
			Prep 1.0	PP 1 · 1 · 0.2	**with**
				Noun 1.0 NP 0.2	**cream**

S	→	NP VP	0.8
S	→	S conj S	0.2
NP	→	Noun	0.2
NP	→	Det Noun	0.4
NP	→	NP PP	0.2
NP	→	NP conj NP	0.2
VP	→	Verb	0.3
VP	→	Verb NP	0.3
VP	→	Verb NP NP	0.1
VP	→	VP PP	0.3
PP	→	Prep NP	1.0
Prep	→	P	1.0
Noun	→	N	1.0
Verb	→	V	1.0

How do we handle flat rules?

S	$\longrightarrow$	NP VP	0.8
S	$\longrightarrow$	S conj S	0.2
NP	$\longrightarrow$	Noun	0.2
NP	$\longrightarrow$	Det Noun	0.4
NP	$\longrightarrow$	NP PP	0.2
NP	$\longrightarrow$	NP conj NP	0.2
VP	$\longrightarrow$	Verb	0.3
VP	$\longrightarrow$	Verb NP	0.3
VP	$\longrightarrow$	Verb NP NP	0.1
VP	$\longrightarrow$	VP PP	0.3
PP	$\longrightarrow$	Prep NP	1.0

S	$\longrightarrow$	**S ConjS**	**0.2**
ConjS	$\longrightarrow$	**conj S**	**1.0**

Binarize each flat rule by adding dummy nonterminals (ConjS),
and setting the probability of the rule with the dummy nonterminal on the LHS to 1

Parser evaluation

Precision and recall

Precision and recall were originally developed as evaluation metrics for information retrieval:

- **Precision:** What percentage of retrieved documents are relevant to the query?
- **Recall**: What percentage of relevant documents were retrieved?

In NLP, they are often used in addition to accuracy:

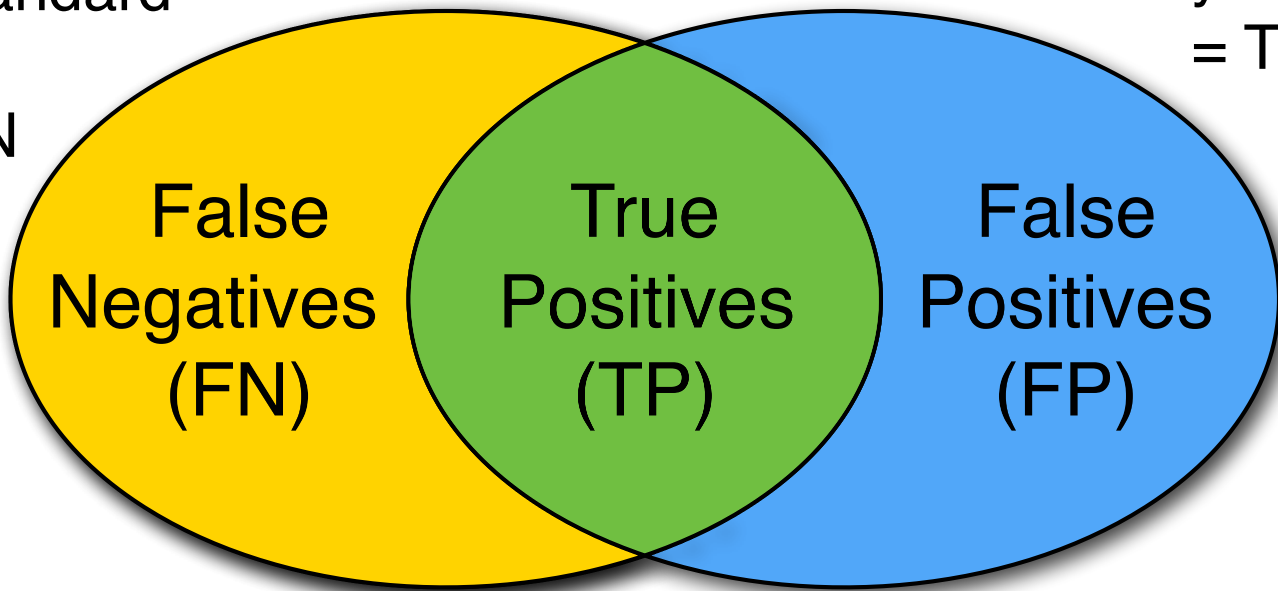
- **Precision:** What percentage of items that were assigned label X do actually have label X in the test data?
- **Recall:** What percentage of items that have label X in the test data were assigned label X by the system?

Particularly useful when there are more than two labels.

True vs. false positives, false negatives

Items labeled X
in the gold standard
('truth')
= TP + FN

Items labeled X
by the system
= TP + FP

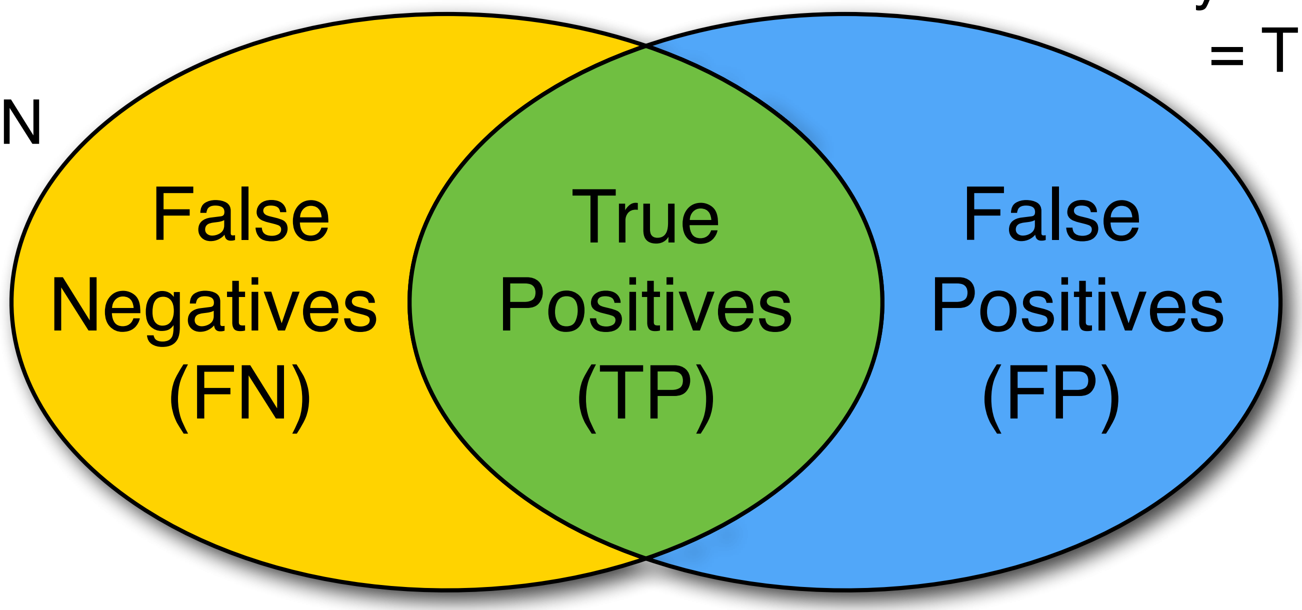


- True positives: Items that were labeled X by the system, and should be labeled X.
- False positives: Items that were labeled X by the system, but should not be labeled X.
- False negatives: Items that were not labeled X by the system, but should be labeled X

Precision, recall, f-measure

Items labeled X
in the gold standard
(‘truth’)
= TP + FN

Items labeled X
by the system
= TP + FP



Precision: P = TP / (TP + FP)

Recall: R = TP / (TP + FN)

F-measure: harmonic mean of precision and recall

$$F = (2 \cdot P \cdot R) / (P + R)$$

Evalb ("Parseval")

Measures recovery of phrase-structure trees.

Labeled: span and label (NP, PP,...) has to be right.

[Earlier variant— unlabeled**:** span of nodes has to be right]

Two aspects of evaluation

Precision: *How many of the predicted nodes are correct?*

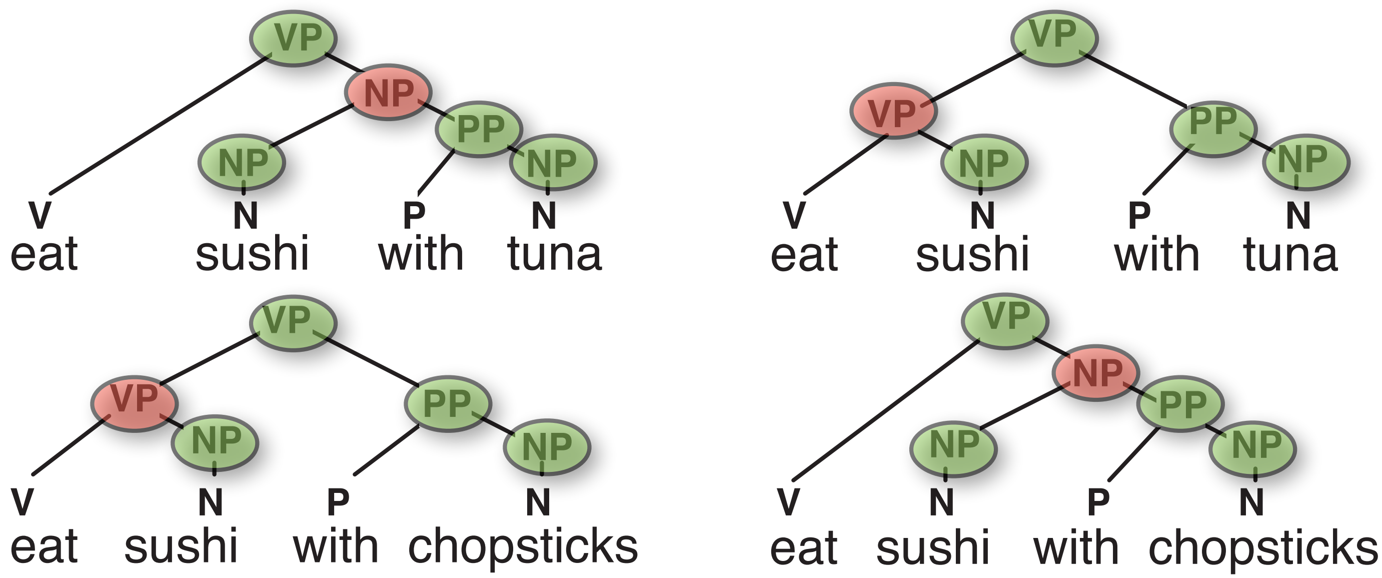
Recall: *How many of the correct nodes were predicted?*

Usually combined into one metric ***(F-measure)****:*

$$
\begin{aligned}
P &= \frac{\#\text{correctly predicted nodes}}{\#\text{predicted nodes}} \\
R &= \frac{\#\text{correctly predicted nodes}}{\#\text{correct nodes}} \\
F &= \frac{2PR}{P+R}
\end{aligned}
$$

Parseval in practice

Gold standard **Parser output**



eat sushi with tuna: Precision: 4/5 Recall: 4/5

eat sushi with chopsticks: Precision: 4/5 Recall: 4/5

Shortcomings of PCFGs

How well can a PCFG model the distribution of trees?

PCFGs make **independence assumptions**:

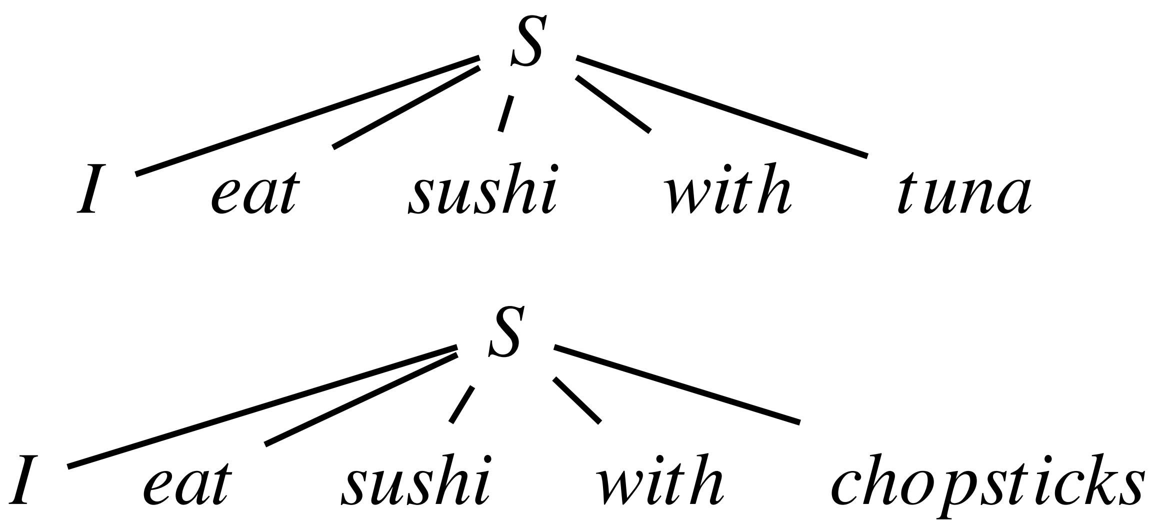
Only the label of a node determines what children it has.

Factors that influence these assumptions:

Shape of the trees:
A corpus with **flat trees** (i.e. few nodes/sentence)
results in a model with few independence assumptions.

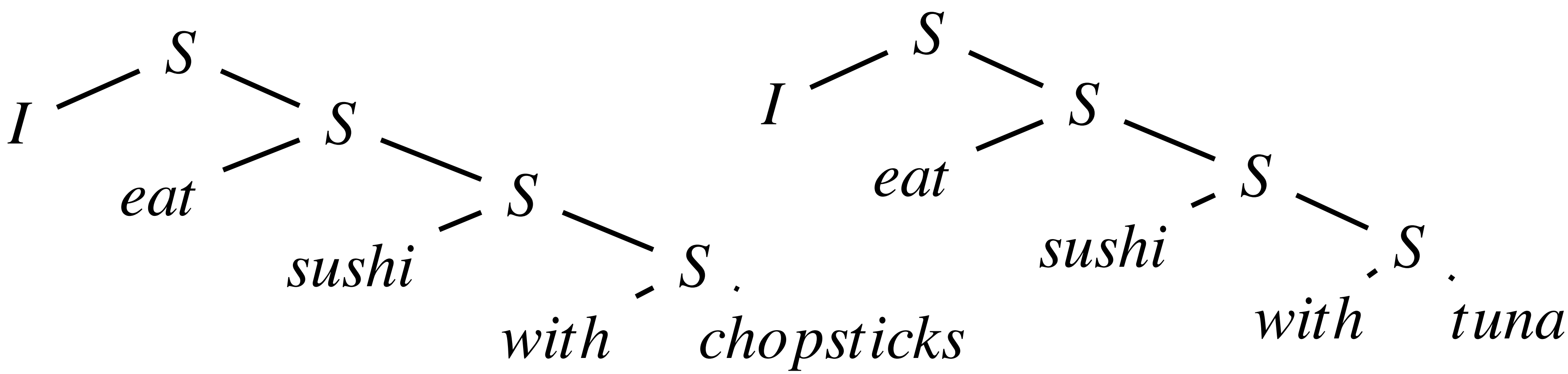
Labeling of the trees:
A corpus with **many node labels** (nonterminals)
results in a model with few independence assumptions.

Example 1: flat trees



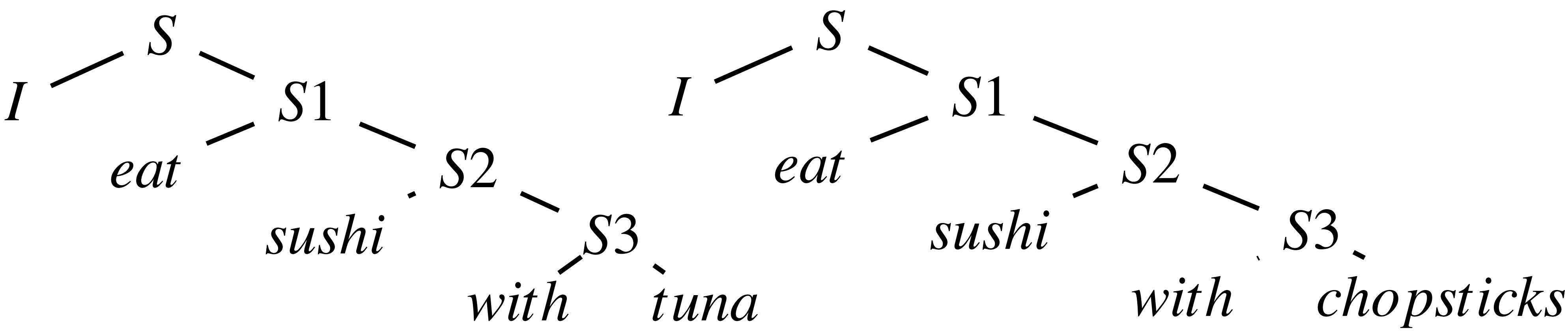
What sentences would a PCFG estimated from this corpus generate?

Example 2: deep trees, few labels



What sentences would a PCFG estimated from this corpus generate?

Example 3: deep trees, many labels



What sentences would a PCFG estimated from this corpus generate?

Aside: Bias/Variance tradeoff

A probability model has low **bias** if it makes few independence assumptions.
⇒ It can capture the structures in the training data.

This typically leads to a more fine-grained partitioning of the training data.

Hence, fewer data points are available to estimate the model parameters.

This increases the **variance** of the model.
⇒ This yields a poor estimate of the distribution.

Penn Treebank parsing

The Penn Treebank

The first publicly available syntactically annotated corpus

Wall Street Journal (50,000 sentences, 1 million words)

also Switchboard, Brown corpus, ATIS

The annotation:

- POS-tagged (Ratnaparkhi's MXPOST)
- Manually annotated with phrase-structure trees
- Richer than standard CFG: *Traces* and other *null elements* used to represent non-local dependencies (designed to allow extraction of predicate-argument structure) [more on this later in the semester]

Standard data set for English parsers

The Treebank label set

48 preterminals (tags):

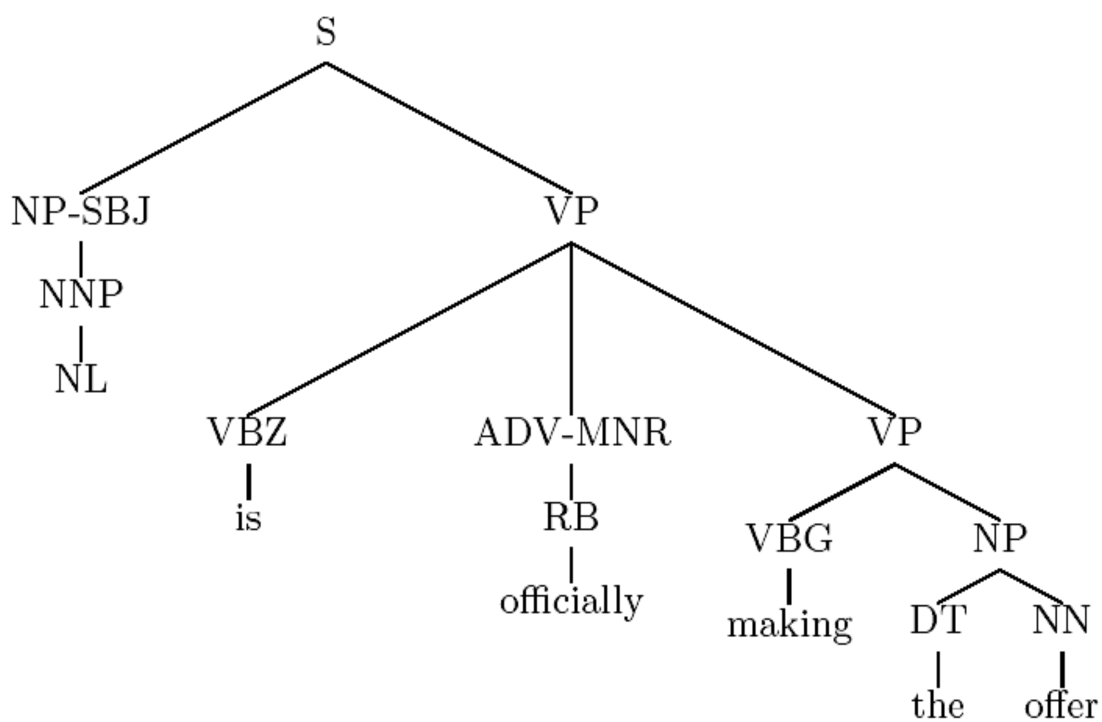
– 36 POS tags, 12 other symbols (punctuation etc.)

– Simplified version of Brown tagset (87 tags)
(cf. Lancaster-Oslo/Bergen (LOB) tag set: 126 tags)

14 nonterminals:

standard inventory (S, NP, VP,...)

A simple example



Relatively flat structures:

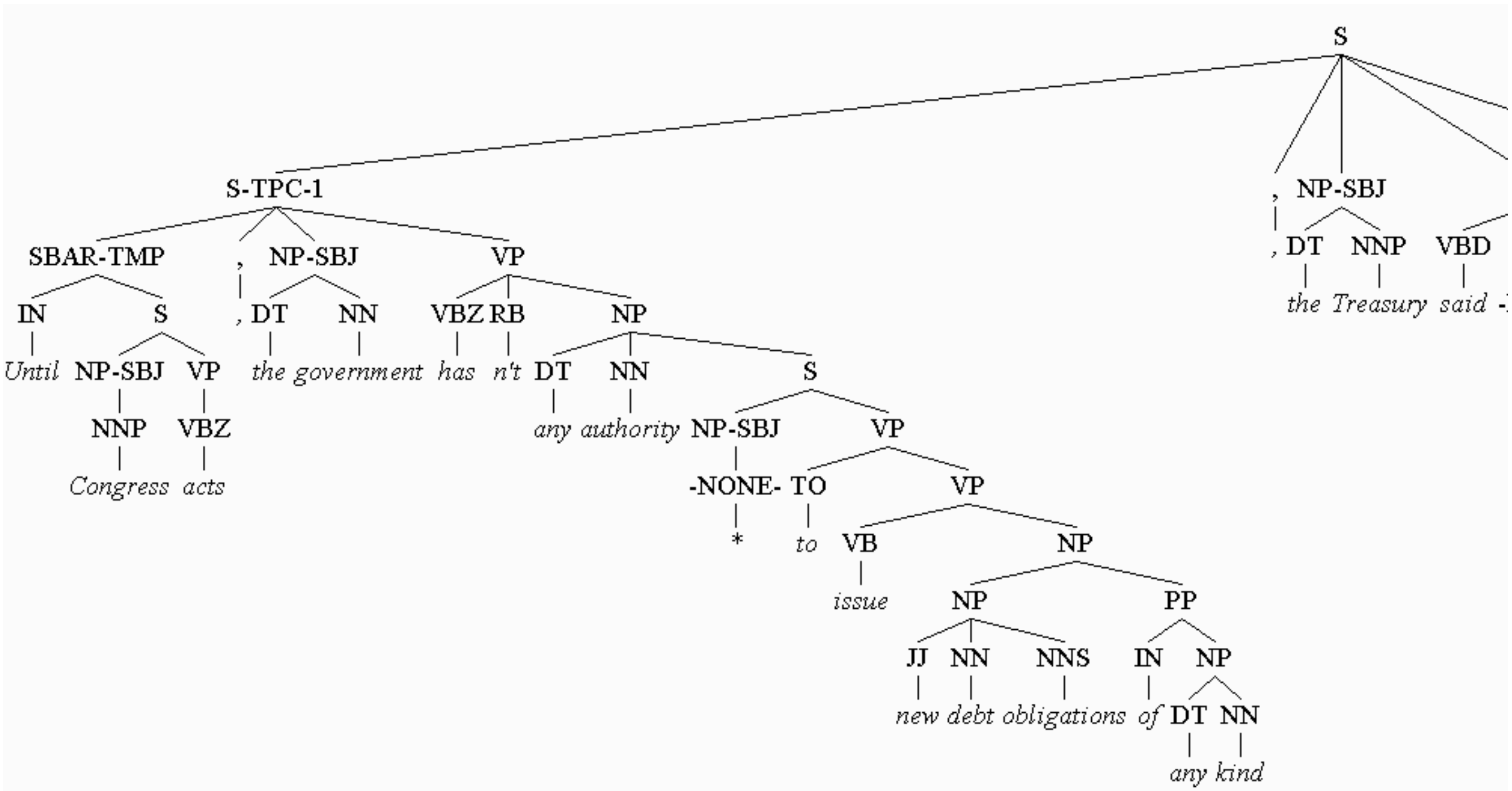
– There is no noun level

– VP arguments and adjuncts appear at the same level

Function tags, e.g. -SBJ (subject), -MNR (manner)

A more realistic (partial) example

Until Congress acts, the government hasn't any authority to issue new debt obligations of any kind, the Treasury said .... .



The Penn Treebank CFG

The Penn Treebank uses very flat rules, e.g.:

NP → DT JJ NN
NP → DT JJ NNS
NP → DT JJ NN NN
NP → DT JJ JJ NN
NP → DT JJ CD NNS
NP → RB DT JJ NN NN
NP → RB DT JJ JJ NNS
NP → DT JJ JJ NNP NNS
NP → DT NNP NNP NNP NNP JJ NN
NP → DT JJ NNP CC JJ JJ NN NNS
NP → RB DT JJS NN NN SBAR
NP → DT VBG JJ NNP NNP CC NNP
NP → DT JJ NNS , NNS CC NN NNS NN
NP → DT JJ JJ VBG NN NNP NNP FW NNP
NP → NP JJ , JJ `` SBAR '' NNS

- Many of these rules appear only once.
- Many of these rules are very similar.
- Can we pool these counts?

PCFGs in practice:

Charniak (1996) *Tree-bank grammars*

How well do PCFGs work on the Penn Treebank?

– Split Treebank into test set (30K words) and training set (300K words).
– Estimate a PCFG from training set.
– Parse test set (with correct POS tags).
– Evaluate unlabeled precision and recall

Sentence Lengths	Average Length	Precision	Recall
2-12	8.7	88.6	91.7
2-16	11.4	85.0	87.7
2-20	13.8	83.5	86.2
2-25	16.3	82.0	84.0
2-30	18.7	80.6	82.5
2-40	21.9	78.8	80.4

Two ways to improve performance

… change the (internal) grammar:

- **Parent annotation/state splits:**
  Not all NPs/VPs/DTs/… are the same.
  It matters where they are in the tree

… change the probability model:

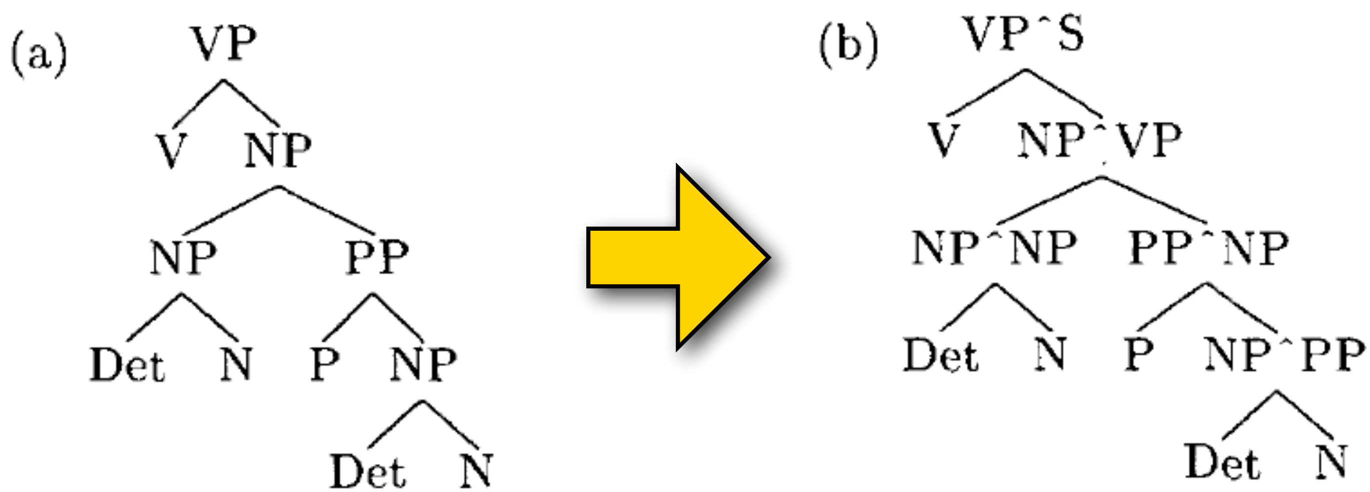
- **Lexicalization:**
  Words matter!
- **Markovization:**
  Generalizing the rules

The parent transformation

PCFGs assume the expansion of any nonterminal is independent of its parent.

- But this is not true: NP subjects more likely to be modified than objects.

We can change the grammar by adding the name of the parent node to each nonterminal



Markov PCFGs (Collins parser)

The RHS of each CFG rule consists of:
one head H_X, *n* left sisters L_i and *m* right sisters R_i:

$$X \longrightarrow \underbrace{L_n \ldots L_1}_{\text{left sisters}} \; H_X \; \underbrace{R_1 \ldots R_m}_{\text{right sisters}}$$

Replace rule probabilities with a generative process:
For each nonterminal X

- generate its head H_X (nonterminal or terminal)
- then generate its left sisters $L_{1..n}$ and a STOP symbol conditioned on H_X
- then generate its right sisters $R_{1...n}$ and a STOP symbol conditioned on H_X

Lexicalization

PCFGs can't distinguish between
"eat sushi with chopsticks" and *"eat sushi with tuna"*.

We need to take words into account!

$P(VP_{eat} \rightarrow VP\ PP_{with\ chopsticks} \mid VP_{eat})$

vs. $P(VP_{eat} \rightarrow VP\ PP_{with\ tuna} \mid VP_{eat})$

Problem: sparse data ($PP_{with\ fatty|white|...\ tuna....}$)
Solution: only take **head words** into account!

Assumption: each constituent has one head word.

Lexicalized PCFGs

At the root (start symbol S), generate the head word of the sentence, w_S, with $P(w_S)$

Lexicalized rule probabilities:
Every nonterminal is lexicalized: X_{w_x}
Condition rules $X_{w_x} \rightarrow \alpha Y \beta$ on the lexicalized LHS X_{w_x}
$P(X_{w_x} \rightarrow \alpha Y \beta \mid X_{w_x})$

Word-word dependencies:
For each nonterminal Y in RHS of a rule $X_{w_x} \rightarrow \alpha Y \beta$, condition w_Y (the head word of Y) on X and w_X:
$P(w_Y \mid Y, X, w_X)$

Dealing with unknown words

A lexicalized PCFG assigns zero probability to any word that does not appear in the training data.

Solution:

Training: Replace rare words in training data with a token ‘UNKNOWN’.

Testing: Replace unseen words with ‘UNKNOWN’

Refining the set of categories

Unlexicalized Parsing (Klein & Manning ’03)

Unlexicalized PCFGs with various transformations of the training data and the model, e.g.:

– Parent annotation (of terminals and nonterminals):

distinguish preposition IN from subordinating conjunction IN etc.

– Add head tag to nonterminals

(e.g. distinguish finite from infinite VPs)

– Add distance features

Accuracy: 86.3 Precision and 85.1 Recall

The Berkeley parser (Petrov et al. ’06, ’07)

Automatically learns refinements of the nonterminals

Accuracy: 90.2 Precision, 89.9 Recall

Summary

The Penn Treebank has a large number of very flat rules.
Accurate parsing requires modifications to the basic PCFG model: refining the nonterminals, relaxing the independence assumptions by including grandparent information, modeling word-word dependencies, etc.

How much of this transfers to other treebanks or languages?