Computational Linguistics

7

CSC 485/2501
Fall 2023

7. Statistical parsing

Gerald Penn
Department of Computer Science, University of Toronto

Reading: Jurafsky & Martin: 5.2–5.5.2, 5.6, 12.4, 14.0–1, 14.3–4, 14.6–7. Bird et al: 8.6.

Statistical parsing 1

- ***General idea:***
  - Assign probabilities to rules in a context-free grammar.
    - Use a likelihood model.
  - Combine probabilities of rules in a tree.
    - Yields likelihood of a parse.
  - The best parse is the *most likely* one.

Statistical parsing 2

- ***Motivations:***
  - Uniform process for attachment decisions.
  - Use lexical preferences in all decisions.

Three general approaches

1. Assign a probability to each rule of grammar, including lexical productions.
   – Parse string of input words with probabilistic rules.
     The can will rust.
2. Assign probabilities only to non-lexical productions.
   – Probabilistically tag input words with syntactic categories using a **part-of-speech tagger**.
   – Consider the pre-terminal syntactic categories to be terminals, parse that string with probabilistic rules.
     Det N Modal Verb.
3. “Supertagging” – parsing as tagging with tree fragments.

Part-of-speech tagging 1

- ***Part-of-speech (PoS) tagging:***
  Given a sequence of words $w_1 \ldots w_n$ (from well-formed text), determine the syntactic category (PoS) C_i of each word.
- *I.e*, the best category sequence $C_1 \ldots C_n$ to assign to the word sequence $w_1 \ldots w_n$.

Most likely

Part-of-speech tagging 2

- Example:

The	*can*	*will*	*rust*
det	**modal verb**	**modal verb**	**noun**
	noun	**noun**	**verb**
	verb	**verb**	

Example from Charniak 1997

Part-of-speech tagging 3

$$P(C_1 \ldots C_n | w_1 \ldots w_n) = \frac{P(C_1 \ldots C_n \wedge w_1 \ldots w_n)}{P(w_1 \ldots w_n)}$$

- We cannot get this probability directly.
- Have to estimate it (through counts).
- Perhaps after first approximating it (by modifying the formula).
- Counts: Need representative corpus.

PoS tagging: Unigram MLE 1

- Look at individual words (***unigrams***):

$$P(C|w) = \frac{P(C \wedge w)}{P(w)}$$

- Maximum likelihood estimator (MLE):

$$P(C|w) = \frac{\boxed{c(w \; is \; C)}}{\boxed{c(w)}}$$

Count in corpus

PoS tagging: Unigram MLE 2

- Problems of MLE:
  - Sparse data.
  - Extreme cases:
    a. Undefined if *w* is not in the corpus.
    b. 0 if *w* does not appear in a particular category.

PoS tagging: Unigram MLE 3

- ***Smoothing*** of formula, e.g.,:

$$P(C|w) \approx \frac{c(w\ is\ C) + \epsilon}{c(w) + \epsilon N}$$

- Give small (non-zero) probability value to unseen events, taken from seen events by *discounting* them.
- Various methods to ensure we still have valid probability distribution.

PoS tagging: Unigram MLE 4

- Just choosing the most frequent PoS for each word yields 90% accuracy in PoS tagging.
- But:
  - Not uniform across words.
  - Accuracy is low (0.9^n) when multiplied over *n* words.
  - No context: *The fly* vs. *I will fly*.
- Need better approximations for

$$P(C_1 \ldots C_n | w_1 \ldots w_n)$$

PoS tagging: Bayesian method

- Use Bayes's rule to rewrite:

$$P(C_1 \ldots C_n | w_1 \ldots w_n) = \frac{\boxed{P(C_1 \ldots C_n)} \times \boxed{P(w_1 \ldots w_n \mid C_1 \ldots C_n)}}{P(w_1 \ldots w_n)}$$

1 2

- For a given word string, we want to maximize this, find most likely $C_1 \ldots C_n$:

$$\operatorname*{argmax}_{C_1 \ldots C_n} P(C_1 \ldots C_n \mid w_1 \ldots w_n)$$

- So just need to maximize the numerator.

Approximating probabilities 1

- Approximate ❶ $P(C_1 \ldots C_n)$ by predicting each category from previous N - 1 categories: an ***N*-gram model**.

Warning: Not the same *n*!!

- Bigram (2-gram) model:

$$P(C_1 \ldots C_n) \approx \prod_{i=1}^{n} P(C_i | C_{i-1})$$

- Posit pseudo-categories START at C_0, and END as C_n. Example:

$$P(\text{A N V N}) \approx P(\text{A}|\text{START}) \cdot P(\text{N}|\text{A}) \cdot P(\text{V}|\text{N}) \cdot P(\text{N}|\text{V}) \cdot P(\text{END}|\text{N})$$

Approximating probabilities 2

- Approximate ❷ $P(w_1 \ldots w_n | C_1 \ldots C_n)$ by assuming that the probability of a word appearing in a category is independent of the words surrounding it.

$$P(w_1 \ldots w_n | C_1 \ldots C_n) \approx \prod_{i=1}^{n} P(w_i | C_i)$$

Lexical generation probabilities

Approximating probabilities 3

- Why is $P(w|C)$ better than $P(C|w)$?
  - $P(C|w)$ is clearly *not* independent of surrounding categories.
  - Lexical generation probability is somewhat more independent.
  - Complete formula for PoS includes bigrams, and so it does capture some context.

Putting it all together

$$
\begin{aligned}
& P(C_1 \ldots C_n \mid w_1 \ldots w_n) \\
= \quad & \frac{P(C_1 \ldots C_n \wedge w_1 \ldots w_n)}{P(w_1 \ldots w_n)} \\
= \quad & \frac{P(C_1 \ldots C_n) \times P(w_1 \ldots w_n \mid C_1 \ldots C_n)}{P(w_1 \ldots w_n)} \\
\propto \quad & P(C_1 \ldots C_n) \times P(w_1 \ldots w_n \mid C_1 \ldots C_n) \\
\approx \quad & \prod_{i=1}^{n} P(C_i \mid C_{i-1}) \times P(w_i \mid C_i) \qquad ❸ \\
= \quad & \boxed{\prod_{i=1}^{n} \frac{c(C_{i-1}C_i)}{c(C_{i-1})} \times \boxed{\frac{c(w_i \textit{ is } C_i)}{c(C_i)}}}
\end{aligned}
$$

Really should use smoothed MLE; *cf* slide 10

MLE for categories not the same as for words; *cf* slide 8

Finding max 1

- Want to find the argmax (most probable) $C_1 \ldots C_n$.
- *Brute force method:* Find all possible sequences of categories and compute P.
- Unnecessary: Our approximation assumes independence:
  - *Category bigrams:* C_i depends only on C_{i-1}. *Lexical generation:* w_i depends only on C_i.
  - Hence we do not need to enumerate all sequences independently.

Finding max 2

- Bigrams:
  Markov model.
  - States are categories and transitions are bigrams.
- Lexical generation probabilities:
  Hidden Markov model.
  - Words are outputs (with given probability) of states.
  - A word could be the output of more than one state.
  - Current state is unknown (“hidden”).

Example

Based on an example in section 7.3 of: Allen, James. *Natural Language Understanding* (2nd ed), 1995, Benjamin Cummings.

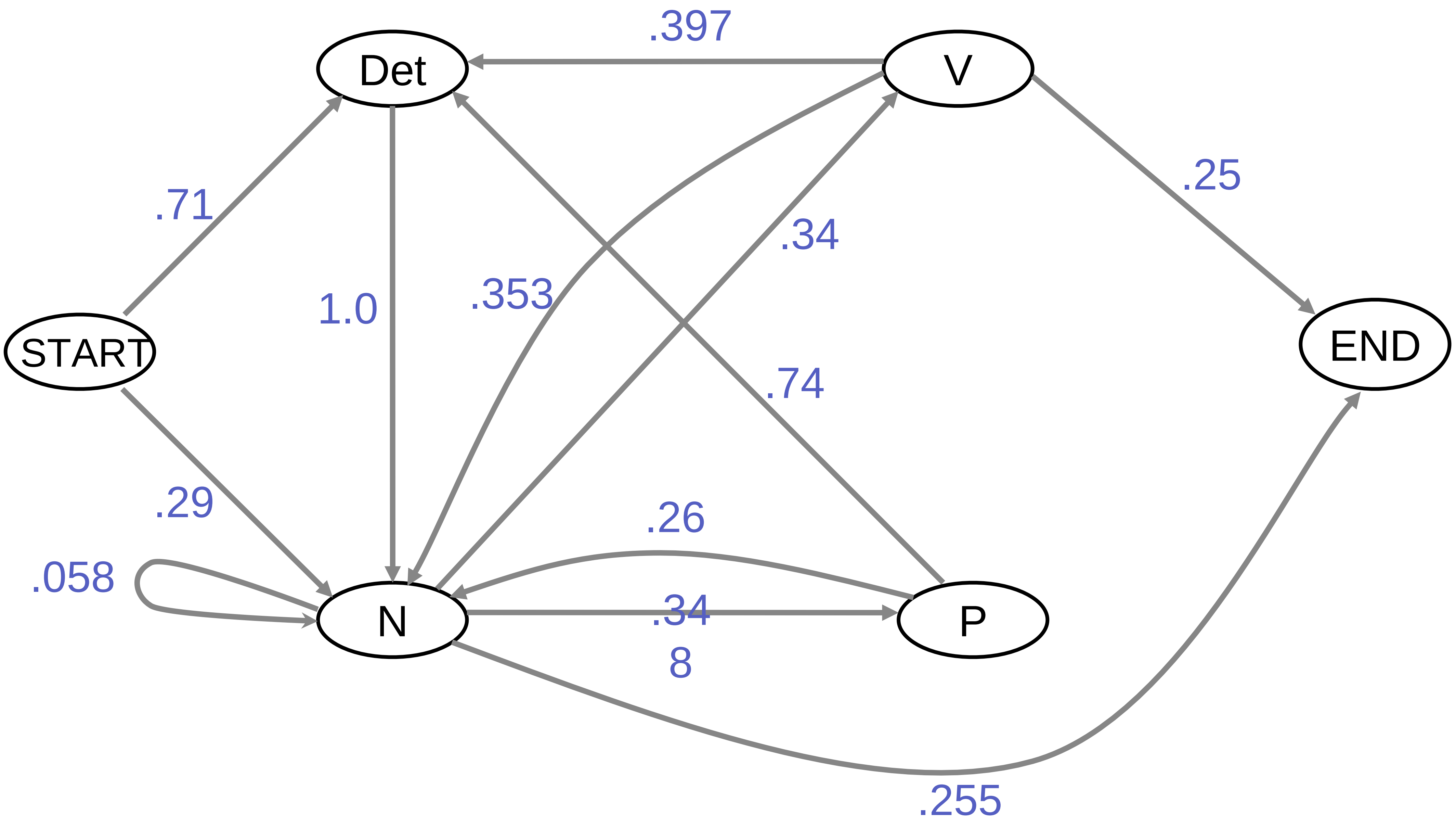
- Artificial corpus of PoS-tagged 300 sentences using only Det, N, V, P.
  - *The flower flowers like a bird.*
    Some birds like a flower with fruit beetles.
    Like flies like flies.
    …
- Some lexical generation probabilities:

$P(the \mid \text{Det}) = .54$	$P(like \mid \text{N}) = .012$	$P(flower \mid \text{N}) = .063$	$P(birds \mid \text{N}) = .076$
$P(a \mid \text{Det}) = .36$	$P(like \mid \text{V}) = .1$	$P(flower \mid \text{V}) = .050$	$P(flies \mid \text{V}) = .076$
$P(a \mid \text{N}) = .001$	$P(like \mid \text{P}) = .068$	$P(flowers \mid \text{N}) = .050$	$P(flies \mid \text{N}) = .025$
⋮	⋮	$P(flowers \mid \text{V}) = .053$	⋮
		⋮	

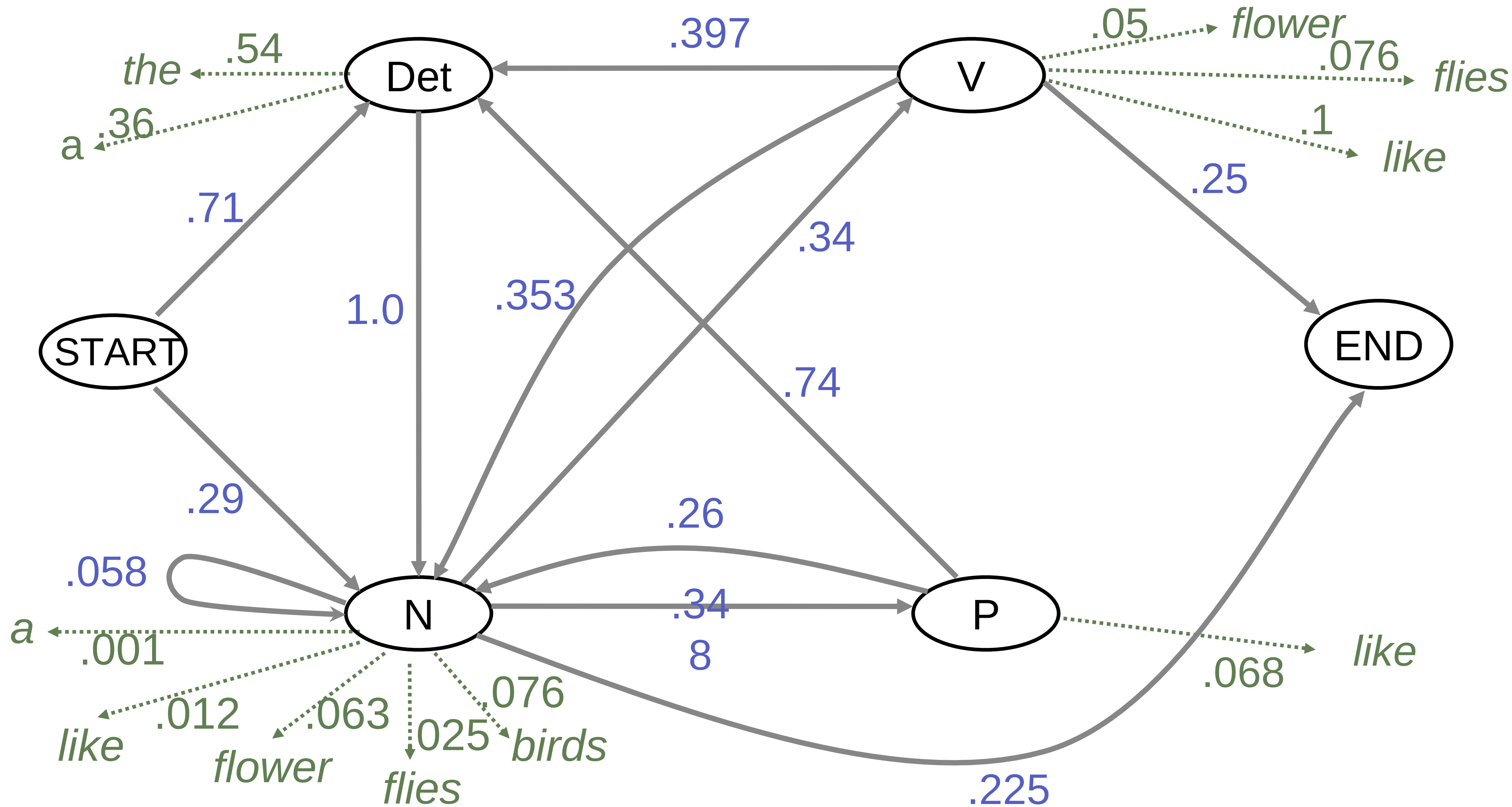
Markov model: Bigram table

Bigram C_{i-1}, C_i	Count C_{i-1}	Count C_{i-1},C_i	$P(C_i \mid C_{i-1})$	Estimate
START, Det	300	213	*P*(Det| START)	0.710
START, N	300	87	*P*(N| START)	0.290
Det, N	558	558	*P*(N|Det)	1.000
N, V	883	300	*P*(V|N)	0.340
N, N	883	51	*P*(N|N)	0.058
N, P	883	307	*P*(P|N)	0.348
N, END	883	225	*P*(END|N)	0.255
V, N	300	106	*P*(N|V)	0.353
V, Det	300	119	*P*(Det|N)	0.397
V, END	300	75	*P*(END|V)	0.250
P, Det	307	226	*P*(Det|P)	0.740
P, N	307	81	*P*(N|P)	0.260

Markov model: Transition probabilities



HMM: Lexical generation probabilities



$P(\textit{the}|\text{Det}) = .54$ $P(\textit{like}|\text{N}) = .012$ $P(\textit{flower}|\text{N}) = .063$ $P(\textit{birds}|\text{N}) = .076$

$P(\textit{a}|\text{Det}) = .36$ $P(\textit{like}|\text{V}) = .1$ $P(\textit{flower}|\text{V}) = .050$ $P(\textit{flies}|\text{V}) = .076$

$P(\textit{a}|\text{N}) = .001$ $P(\textit{like}|\text{P}) = .068$ $P(\textit{flowers}|\text{N}) = .050$ $P(\textit{flies}|\text{N}) = .025$

⋮ ⋮ $P(\textit{flowers}|\text{V}) = .053$ ⋮

Hidden Markov models 1

- Given the observed output, we want to find the most likely path through the model.

The	*can*	*will*	*rust*
det	modal verb	**modal verb**	noun
	noun	noun	**verb**
	verb	verb	

Hidden Markov models 2

- At any state in an HMM, how you got there is irrelevant to computing the next transition.
  - So, just need to remember the best path and probability up to that point.
  - Define $\phi(C_{i-1})$ as the probability of the best sequence up to state C_{i-1}.
- Then find C_i that maximizes
  $\varphi(C_{i-1}) \times \boxed{P(C_i|C_{i-1}) \times P(w|C_i)}$ ❸ from slide 17

Viterbi Algorithm

- Given an HMM and an observation O of its output, finds the most probable sequence S of states that produced O.
  - O = words of sentence, S = PoS tags of sentence
- Parameters of HMM based on large training corpus.
- Then find C_i that maximizes
  $\varphi(C_{i-1}) \times P(C_i|C_{i-1}) \times P(w|C_i)$

  $\beta_i = C_{i-1}$ [*backtrace*]

Baum-Welch Algorithm

- Given an HMM M and an observation O, adjust the parameters of M to improve the probability P(O).
  - O = words of sentence, M = $\langle \pi, A, B \rangle$
- This is an instance of Expectation-Maximization (EM).

Statistical chart parsing 1

- Consider tags as terminals (*i.e.*, use a PoS tagger to pre-process input texts).

  Det N Modal Verb.

- For probability of each grammar rule, use MLE.
- Probabilities derived from hand-parsed corpora (treebanks).
  - Count frequency of use *c* of each rule $C \longrightarrow \alpha$, for each non-terminal *C* and each different RHS α.

What are some problems with this approach?

Statistical chart parsing 2

- MLE probability of rules:
  - For each rule $C \rightarrow \alpha$:

$$P(C \rightarrow \alpha | C) = \frac{c(C \rightarrow \alpha)}{\sum_\beta c(C \rightarrow \beta)} = \frac{c(C \rightarrow \alpha)}{c(C)}$$ 4

- Takes no account of the context of use of a rule: ***independence assumption.***
- ***Source-normalized:*** assumes a top-down generative process.
- NLTK's pchart demo doesn't POS-tag first (words are generated top-down), and it shows P(t) rather than P(t|s). Why?

```
>>> import nltk
>>> nltk.parse.pchart.demo()

  1: I saw John with my telescope
     <Grammar with 17 productions>

  2: the boy saw Jack with Bob under the table with a telescope
     <Grammar with 23 productions>

Which demo (1-2)? 1

s: I saw John with my telescope
parser: <nltk.parse.pchart.InsideChartParser object at 0x7f61288f3290>
grammar: Grammar with 17 productions (start state = S)
    S -> NP VP [1.0]
    NP -> Det N [0.5]
    NP -> NP PP [0.25]
    NP -> 'John' [0.1]
    NP -> 'I' [0.15]
    Det -> 'the' [0.8]
    Det -> 'my' [0.2]
    N -> 'man' [0.5]
    N -> 'telescope' [0.5]
    VP -> VP PP [0.1]
    VP -> V NP [0.7]
    VP -> V [0.2]
    V -> 'ate' [0.35]
    V -> 'saw' [0.65]
    PP -> P NP [1.0]
    P -> 'with' [0.61]
    P -> 'under' [0.39]
```

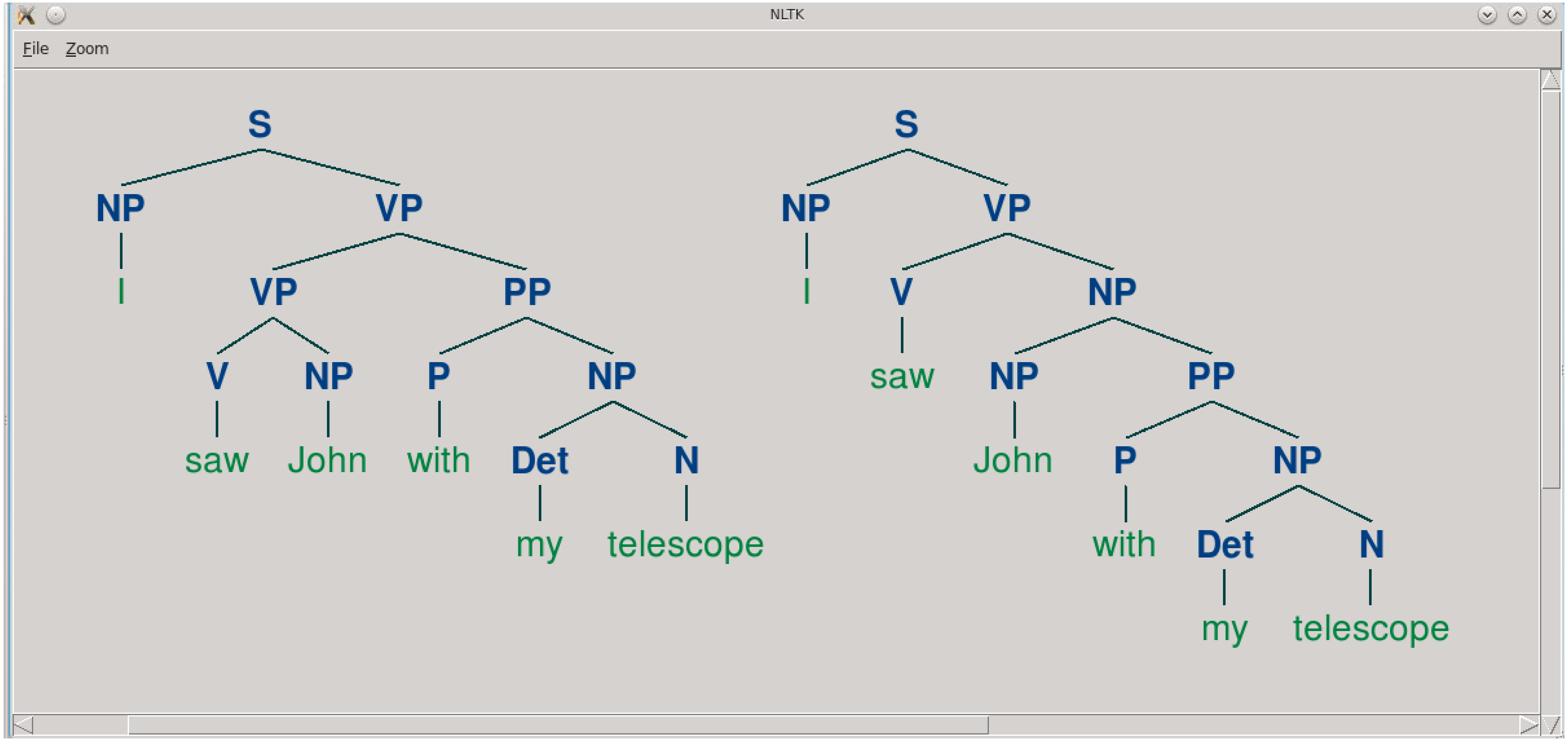
```
|[-] . . . . . .| [0:1] 'I'                                   [1.0]
|. [-] . . . . .| [1:2] 'saw'                                 [1.0]
|. . [-] . . . .| [2:3] 'John'                                [1.0]
|. . . [-] . . .| [3:4] 'with'                                [1.0]
|. . . . [-] . .| [4:5] 'my'                                  [1.0]
|. . . . . . [-]| [5:6] 'telescope'                           [1.0]
|. . . . . . [-]| [5:6] 'telescope'                           [1.0]
|. . . . [-] . .| [4:5] 'my'                                  [1.0]
|. . . [-] . . .| [3:4] 'with'                                [1.0]
|. . [-] . . . .| [2:3] 'John'                                [1.0]
|. [-] . . . . .| [1:2] 'saw'                                 [1.0]
|[-] . . . . . .| [0:1] 'I'                                   [1.0]
|. [-] . . . . .| [1:2] V  -> 'saw' *                         [0.65]
|. > . . . . . .| [1:1] VP -> * V NP                          [0.7]
|. > . . . . . .| [1:1] V  -> * 'saw'                         [0.65]
|. . . [-] . . .| [3:4] P  -> 'with' *                        [0.61]
|. . . > . . . .| [3:3] PP -> * P NP                          [1.0]
|. . . [-> . . .| [3:4] PP -> P * NP                          [0.61]
|. . . > . . . .| [3:3] P  -> * 'with'                        [0.61]
|. . . . . . [-]| [5:6] N  -> 'telescope' *                   [0.5]
|. . . . . . > .| [5:5] N  -> * 'telescope'                   [0.5]
|. [-> . . . . .| [1:2] VP -> V * NP                          [0.455]
|. > . . . . . .| [1:1] VP -> * V                             [0.2]
|. . . . [-] . .| [4:5] Det -> 'my' *                         [0.2]
|. . . . > . . .| [4:4] NP -> * Det N                         [0.5]
|. . . . > . . .| [4:4] Det -> * 'my'                         [0.2]
```

:
:

⋮

```
|. . . . > . .| [4:4] S  -> * NP VP          [1.0]
|. . . . > . .| [4:4] NP -> * NP PP          [0.25]
|. . . . [--->| [4:6] S  -> NP * VP          [0.05]
|. [---] . . .| [1:3] VP -> V NP *           [0.0455]
|[-> . . . . .| [0:1] NP -> NP * PP          [0.0375]
|. . . [-----]| [3:6] PP -> P NP *           [0.0305]
|. . [-> . . .| [2:3] NP -> NP * PP          [0.025]
|[---] . . . .| [0:2] S  -> NP VP *          [0.0195]
|. [-> . . . .| [1:2] VP -> VP * PP          [0.013]
|. . . . [--->| [4:6] NP -> NP * PP          [0.0125]
|[-----] . . .| [0:3] S  -> NP VP *          [0.006825]
|. [---> . . .| [1:3] VP -> VP * PP          [0.00455]
|. . [-------]| [2:6] NP -> NP PP *          [0.0007625]
|. . [------->| [2:6] S  -> NP * VP          [0.0007625]
|. [---------]| [1:6] VP -> V NP *           [0.0003469375]
|. . [------->| [2:6] NP -> NP * PP          [0.000190625]
|. [---------]| [1:6] VP -> VP PP *          [0.000138775]
|[===========]| [0:6] S  -> NP VP *          [5.2040625e-05]
|. [--------->| [1:6] VP -> VP * PP          [3.469375e-05]
|[===========]| [0:6] S  -> NP VP *          [2.081625e-05]
|. [--------->| [1:6] VP -> VP * PP          [1.38775e-05]
```

```
Draw parses (y/n)? y
   please wait...
```



```
Print parses (y/n)? y
 (S
  (NP I)
  (VP
    (VP (V saw) (NP John))
    (PP (P with) (NP (Det my) (N telescope))))) [2.081625e-05]
(S
  (NP I)
  (VP
    (V saw)
    (NP
      (NP John)
      (PP (P with) (NP (Det my) (N telescope)))))) [5.2040625e-05]
```

Statistical chart parsing 3

- In this view of chart parsing, probability of chart entries is relatively simple to calculate. For completed constituents, maximize over $C_1, \ldots, C_n$ (like Viterbi):

$$\begin{aligned} P(e_0) &= P(C_0 \to C_1 \ldots C_n \mid C_0) \times P(e_1) \times \cdots \times P(e_n) \\ &= P(C_0 \to C_1 \ldots C_n \mid C_0) \times \prod_{i=1}^{n} P(e_i) \qquad ❺ \end{aligned}$$

e_0 is the entry for current constituent, of category C_0;

$e_1 \ldots e_n$ are chart entries for $C_1 \ldots C_n$ in the RHS of the rule.

NB: Unlike for PoS tagging above, the C_i are not necessarily lexical categories.

Statistical chart parsing 4

- Consider a complete parse tree, *t*, with root label S.
- Recasting ❺, *t* has the probability:

$$P(t) = P(S) * \Pi_n P(rule(n)|cat(n)) \quad ❻$$

where *n* ranges over all nodes in the tree *t*;
rule(n) is the rule used for *n*;
cat(*n*) is the category of *n*.

- *P(S) = 1!*
- “Bottoms out” at lexical categories.
- Note that we’re parsing bottom-up, but the generative model “thinks” top-down regardless.

Inside-Outside Algorithm

- EM for PCFGs: maximum likelihood estimates on an annotated corpus can be improved to increase the likelihood of a different, unannotated corpus
- Step 1: parse the unannotated corpus using the MLE parameters.
- Step 2: adjust the parameters according to the expected relative frequencies of different rules in the parse trees obtained in Step 1:
  - $\dot{p}(A \rightarrow B\ C) = \mu(A \rightarrow B\ C) / Z$
  - $\dot{p}(A \rightarrow w) = \mu(A \rightarrow w) / Z$

Inside-Outside Algorithm 2

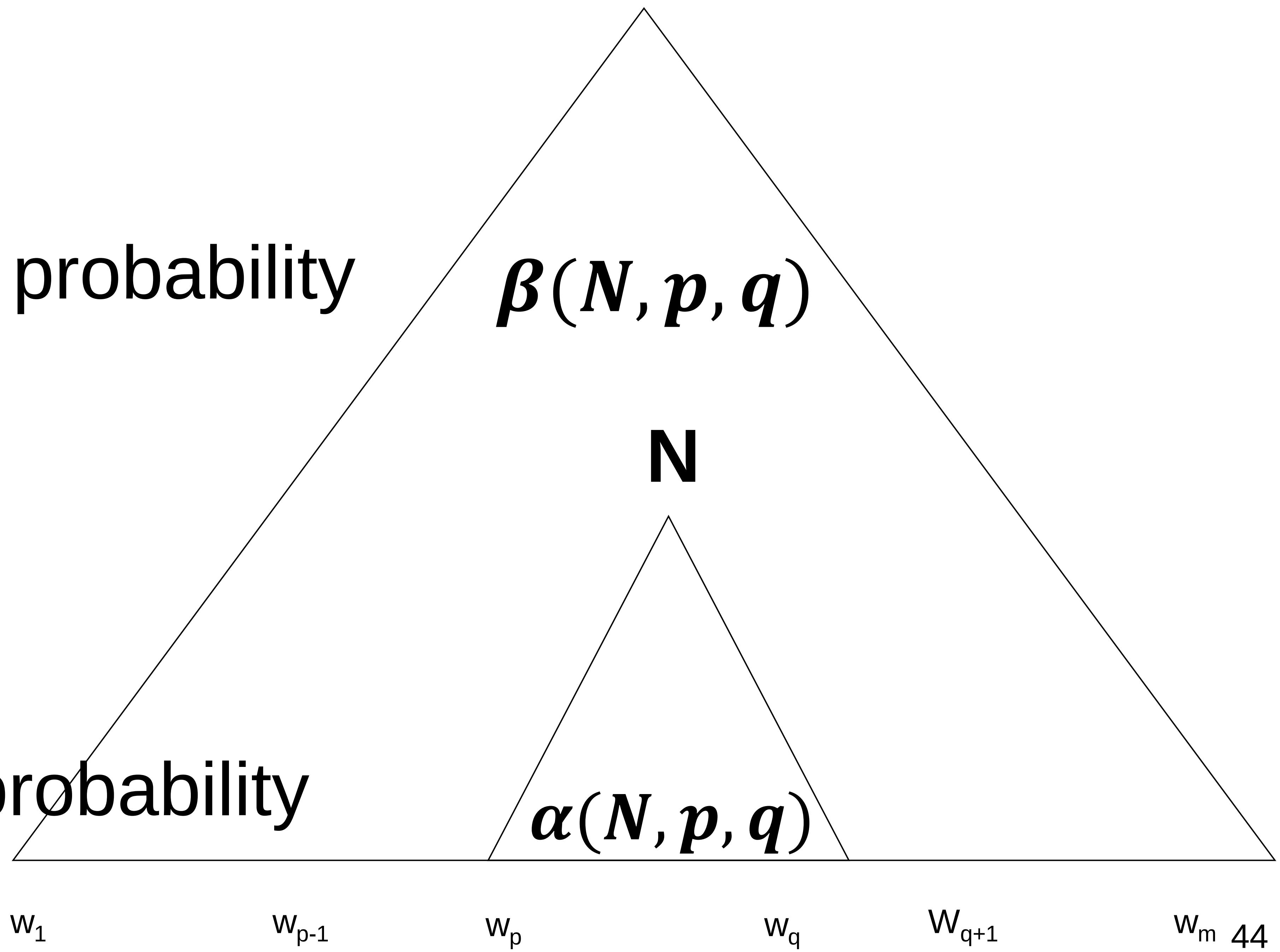
- $\mu(A \rightarrow BC) = \sum_{\{i,k,j\}} \mu(A \rightarrow BC, i, k, j)$
- $\mu(A \rightarrow w) = \sum_i \mu(A, i)\delta_i(w)$

where we now count having seen an A from i to j, a B from i to k, and a C from k to j,

…or an A at location i, where there appears the word w.

Inside-Outside Algorithm 3

- We can define these position-specific μ's in terms of:
  - outside probability
  - inside probability



Inside-Outside Algorithm 4

- $\mu(A \rightarrow BC, i, k, j) =$
  $p(A \rightarrow BC)\ \beta(A, i, j)\, \alpha(B, i, k)\, \alpha(C, k+1, j)$
- $\mu(A, i) = \mu(A, i, i)$
- $\mu(A, i, j) = \alpha(A, i, j)\ \beta(A, i, j)$
- $Z = \alpha(S, 1, n)$

There are also very terse, recursive formulations of α and β that are amenable to dynamic programming.

Statistical chart parsing 5

- But just like non-statistical chart parsers, this one only answers ‘yes’ or ‘no’ (with a probability) in polynomial time:
  - It’s not supposed to matter how we got each constituent. Just the non-terminal label and the span are all that should matter.
- There might be exponentially many trees in this formulation.
- And we’re not calculating the probability that the input is a sentence – this is only the probability of one interpretation (tree).

Evaluation 1

- Evaluation method:
  - *Train* on part of a parsed corpus.
    (*I.e.*, gather rules and statistics.)
  - Test on a different part of the corpus.
    - Development test: early stopping, metaparameters
    - Evaluation test: evaluate (and then done)
- In one sense, the best evaluation of a method like this would be data likelihood, but since we're scoring trees instead of strings, it's difficult to defend any sort of intuition about the numbers assigned to them.

Evaluation 2

- **Evaluation:** PARSEVAL measures compare parser output to known correct parse:
  - Labelled **precision**, labelled **recall**.

Fraction of constituents in output that are correct.

Fraction of correct constituents in output.

  - ***F*-measure** = harmonic mean of precision and recall = $2PR / (P + R)$

Evaluation 3

- **Evaluation:** PARSEVAL measures compare parser output to known correct parse:
  - Penalize for **cross-brackets** per sentence: Constituents in output that overlap two (or more) correct ones; e.g., [[A B] C] for [A [B C]].

    [[*Nadia*] [[*smelled*] [*the eggplant*]]]

    [[[*Nadia*] [*smelled*]] [*the eggplant*]]

  The labels on the subtrees aren't necessary for this one.

Evaluation 4

- PARSEVAL is a *classifier accuracy* score – much more extensional. All that matters is the right answer at the end.
- But that still means that we can look at parts of the right answer.
- Can get ~75% labelled precision, recall, and *F* with above methods.

Improving statistical parsing

- Problem: Probabilities are based only on structures and categories:

$$P(C \rightarrow \alpha | C) = \frac{c(C \rightarrow \alpha)}{\sum_{\beta} c(C \rightarrow \beta)} = \frac{c(C \rightarrow \alpha)}{c(C)} \quad \text{❹}$$

- But actual words strongly condition which rule is used (*cf* Ratnaparkhi).
- Improve results by conditioning on more factors, including words. Think *semantics* – the words themselves give us a little bit of access to this.

Lexicalized grammars 1

- ***Head*** of a phrase: its central or key word.
  - The noun of an NP, the preposition of a PP, etc.
- ***Lexicalized*** grammar: Refine the grammar so that rules take heads of phrases into account — the actual words.
  - BEFORE: Rule for NP.
    AFTER: Rules for NP-whose-head-is-*aardvark*, NP-whose-head-is-*abacus*, …, NP-whose-head-is-*zymurgy*.
- And similarly for VP, PP, etc.

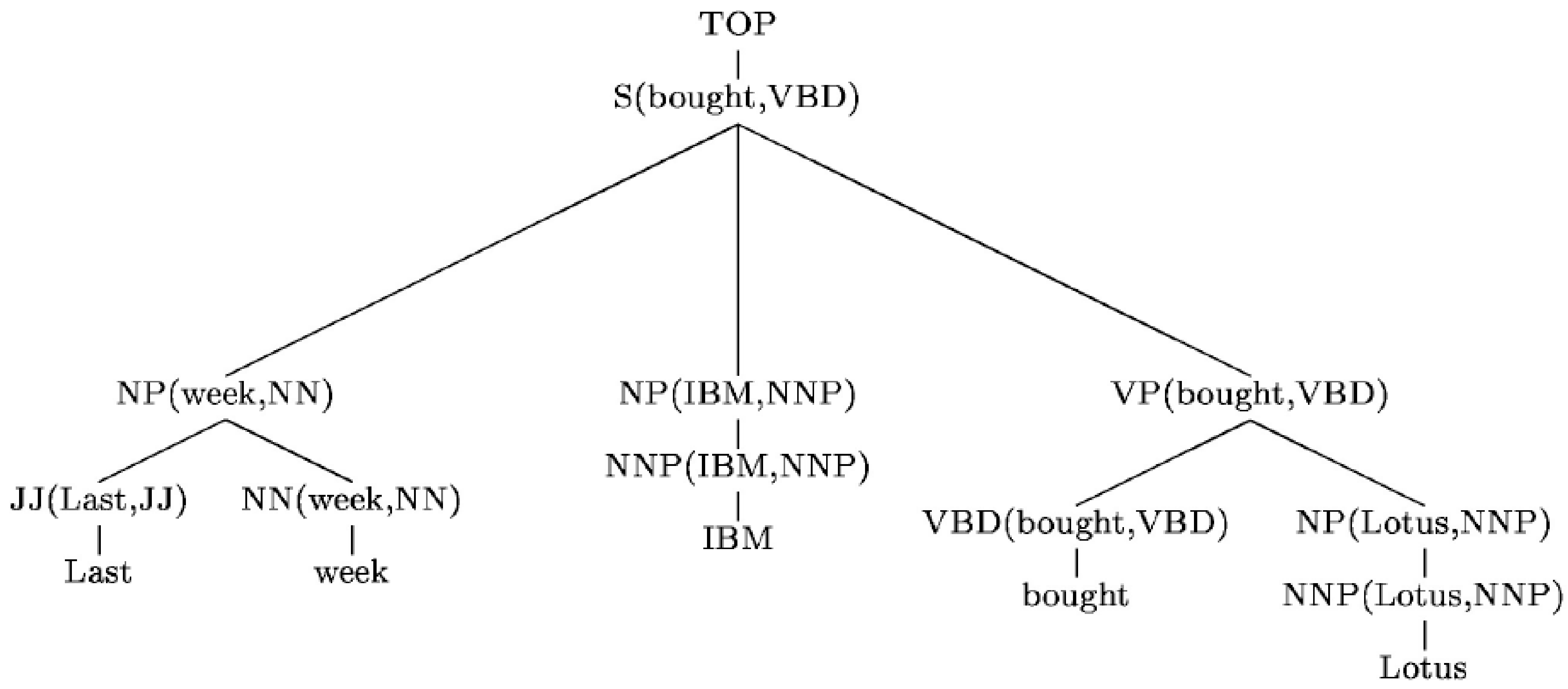
Lexicalized grammars 2

- Notation: *cat*(*head,tag*) for constituent category *cat* headed by *head* with part-of-speech *tag*.
  - e.g., NP(*aardvark*,NN), PP(*without*,IN)

NP-whose-head-is-the-NN-*aardvark*

PP-whose-head-is-the-IN-*without*

A lexicalized grammar



TOP → S(*bought*,VBD)
S(*bought*,VBD) → NP(*week*,NN) NP(*IBM*,NNP) VP(*bought*,VBD)
NP(*week*,NN) → JJ(*Last*,JJ) NN(*week*,NN)
NP(*IBM*,NNP) → NNP(*IBM*,NNP)
VP(*bought*,VBD) → VBD(*bought*,VBD) NP(*Lotus*,NNP)
NP(*Lotus*,NNP) → NNP(*Lotus*,NNP)

Lexical Rules:

JJ(*Last*,JJ) → *Last*
NN(*week*,NN) → *week*
NNP(*IBM*,NNP) → *IBM*
VBD(*bought*,VBD) → *bought*
NNP(*Lotus*,NNP) → *Lotus*

Michael Collins. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4), 2003, 589–637.

Lexicalized grammars 3

- Number of rules and categories explodes, but no theoretical change in parsing process (whether statistical or not).
- But far too specific for practical use; each is too rarely used to determine its probability.
- Need something more than regular (unlexicalized) rules and less than complete lexicalization …
- … perhaps we should change the process after all.

Lexicalized parsing 1

Starting from unlexicalized rules:

- ***1. Lexicalization:*** Consider the head word of each node, not just its category:
- $P(t) = P(S) * \Pi_n P(rule(n)|head(n))$

Replaces ❻ from slide 41

where *head*(*n*) is the PoS-tagged head word of node *n*.

- Needs finer-grained probabilities:
  - e.g., probability that rule *r* is used, given we have an NP whose head is the noun *deficit*.

Lexicalized parsing 2

- ***2. Head and parent:*** Condition on the head and the head of the parent node in the tree:

$$P(\text{Sentence, Tree})$$
$$= \prod_{n \in \text{Tree}} P(rule(n) \mid head(n)) \times P(head(n) \mid head(parent(n)))$$

e.g., probability of rule *r* given that head is the noun *deficit*.

e.g., probability that head is the noun *deficit*,
given that parent phrase's head is the verb *report*.

Effects on parsing

- Lexical information introduces *context* into CFG.
- Grammar is larger.
- Potential problems of sparse data.
  - Possible solutions: Smoothing; back-off estimates.

If you don’t have data for a fine-grained situation, use data from a coarser-grained situation that it’s contained in.

Bikel’s 2004 intepretation

- Can condition on *any* information available in generating the tree.
- ***Basic idea:*** Avoid sparseness of lexicalization by decomposing rules.
  - Make plausible independence assumptions.
  - Break rules down into small steps (small number of parameters).
  - Each rule still parameterized with word/PoS pair:

    S(*bought*,VBD) → NP(*week*,NN) NP(*IBM*,NNP) VP(*bought*,VBD)

Michael Collins. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4), 2003, 589–637.

Collins's "model 1" 1

- ***Lexical Rules,*** with probability 1:
  tag(*word, tag*) → *word*
- ***Internal Rules,*** with treebank-based probabilities. Separate terminals to the left and right of the head; generate one at a time:

$$X \rightarrow L_n L_{n-1} \ldots L_1 \, H \, R_1 \, \ldots R_{m-1} \, R_m \quad (n, m \geq 0)$$

X, L_i, H, and R_i all have the form *cat*(*head,tag*).
Notation: Italic lowercase symbol for (*head,tag*):

$$X(x) \rightarrow L_n(l_n) L_{n-1}(l_{n-1}) \ldots L_1(l_1) \, H(h) \, R_1(r_1) \, \ldots R_{m-1}(r_{m-1}) \, R_m(r_m)$$

Collins's "model 1" 2

- Assume there are additional L_{n+1} and R_{m+1} representing phrase boundaries ("STOP").
- Example:
  S(*bought*,VBD) → NP(*week*,NN) NP(*IBM*,NNP) VP(*bought*,VBD)
  $n = 2$, $m = 0$ (two constituents on the left of the head, zero on the right).
  X = S, H = VP, L_1 = NP, L_2 = NP, L_3 = STOP, R_1 = STOP.
  h = (*bought*,VBD), l_1 = (*IBM*,NNP), l_2 = (*week*,NN).
- Distinguish probabilities of heads P_h, of left constituents P_l, and of right constituents P_r.

Probabilities of internal rules 1

$$
\begin{aligned}
& P(\mathrm{X}(h)) \\
& \quad = \quad P(\mathrm{L}_{n+1}(l_{n+1})\mathrm{L}_n(l_n)\ \ldots \mathrm{L}_1(l_1)\ \mathrm{H}(h)\ \mathrm{R}_1(r_1)\ \ldots\ \mathrm{R}_m(r_m)\ \mathrm{R}_{m+1}(r_{m+1}) \mid \mathrm{X}, h) \\
& \quad = \quad P_h(\mathrm{H} \mid \mathrm{X}, h) \\
& \qquad\quad \times \prod_{i=1}^{n+1} P_l(\mathrm{L}_i(l_i) \mid \mathrm{L}_1(l_1) \ldots \mathrm{L}_{i-1}(l_{i-1}), \mathrm{X}, h, \mathrm{H}) \\
& \qquad\quad \times \prod_{j=1}^{m+1} P_r(\mathrm{R}_j(r_j) \mid \mathrm{L}_1(l_1) \ldots \mathrm{L}_n(l_n), \mathrm{R}_1(r_1) \ldots \mathrm{R}_{j-1}(r_{j-1}), \mathrm{X}, h, \mathrm{H}) \\
& \quad \approx \quad P_h(\mathrm{H} \mid \mathrm{X}, h) \times \prod_{i=1}^{n+1} P_l(\mathrm{L}_i(l_i) \mid \mathrm{X}, h, \mathrm{H}) \times \prod_{j=1}^{m+1} P_r(\mathrm{R}_j(r_j) \mid \mathrm{X}, h, \mathrm{H})
\end{aligned}
$$

Generate head constituent

Generate left modifiers (stop at STOP)

Generate right modifiers (stop at STOP)

By independence assumption

Probabilities of internal rules 2

Example:

$P($ S(*bought*,VBD) $\rightarrow$ NP(*week*,NN) NP(*IBM*,NNP) VP(*bought*,VBD) $)$

$\approx P_h$(VP | S, *bought*,VBD)
$\times\, P_l$(NP(*IBM*,NNP) | S, *bought*,VBD, VP)
$\times\, P_l$(NP(*week*,NN) | S, *bought*,VBD, VP)
$\times\, P_l$(STOP | S, *bought*,VBD, VP)
$\times\, P_r$(STOP | S, *bought*,VBD, VP)

Generate head constituent

Generate left modifiers

Generate right modifiers

Adding other dependencies

- (Badly-named) “distance measure” to capture properties of attachment relevant to current modifier.
  - $P_l(L_i(l_i) \mid X, h, H)$ becomes
    $P_l(L_i(l_i) \mid X, h, H, distance_l(i-1))$
    and analogously on the right.
  - The value of *distance*$_x$ is actually a pair of Boolean random variables:
    - Is string 1..(*i* – 1) of length 0?
      i.e., is attachment of modifier *i* to the head?
    - Does string 1..(*i* – 1) contain a verb?
      i.e., is attachment of modifier *i* crossing a verb?

Collins's "model 1" 4

- Backs off …
    - to tag probability when no data for specific word;
    - to complete non-lexicalization when necessary.

Collins's Models 2 and 3

- *Model 2:* Add verb subcategorization and argument/adjunct distinction.
- *Model 3:* Integrate gaps and trace identification into model.
  - Especially important with addition of subcategorization.

Results and conclusions 1

- Model 2 outperforms Model 1.
- Model 3: Similar performance, but identifies traces too.
- Model 2 performs best overall:
  - LP = 89.6, LR = 89.9 [sentences ≤ 100 words].
  - LP = 90.1, LR = 90.4 [sentences ≤ 40 words].
- Rich information improves parsing performance.

Results and conclusions 2

- **Strengths:**
  - Incorporation of lexical and other linguistic information.
  - Competitive results.
- **Weaknesses:**
  - Supervised training.
  - Performance tightly linked to particular type of corpus used.

Results and conclusions 3

- **Importance to CL:**
  - High-performance parser showing benefits of lexicalization and linguistic information.
  - Publicly available, widely used in research.
  - There was some initial hope that it would make language models better, but that didn't pan out.
  - But it was fairly successful at giving us some access to semantics, i.e. language modelling makes parsing better.