Grammars, graphs and automata

Mark Johnson

Brown University

ESSLLI 2005

slides available from http://cog.brown.edu/~mj

High-level overview

- Probability distributions and graphical models
- (Probabilistic) finite state machines and context-free grammars
  - computation (dynamic programming)
  - estimation
- Log-linear models
  - stochastic unification-based grammars
  - reranking parsing
- Weighted CFGs and proper PCFGs

Topics

- Graphical models and Bayes networks
- (Hidden) Markov models
- (Probabilistic) context-free grammars and finite-state machines
- Computation with and estimation of PCFGs
- Lexicalized and bi-lexicalized PCFGs
- Non-local dependencies and log-linear models
- Features in reranking parsing
- Stochastic unification-based grammar
- Weighted CFGs and proper PCFGs

What is computational linguistics?

Computational linguistics studies the *computational processes* involved in *language production, comprehension and acquisition*.

- assumption that language is *inherently computational*
- scientific side:
  - modeling human performance (computational psycholinguistics)
  - understanding how it can be done at all
- technological applications:
  - speech recognition
  - information extraction (who did what to whom) and question answering
  - machine translation (translation by computer)

(Some of the) problems in modeling language

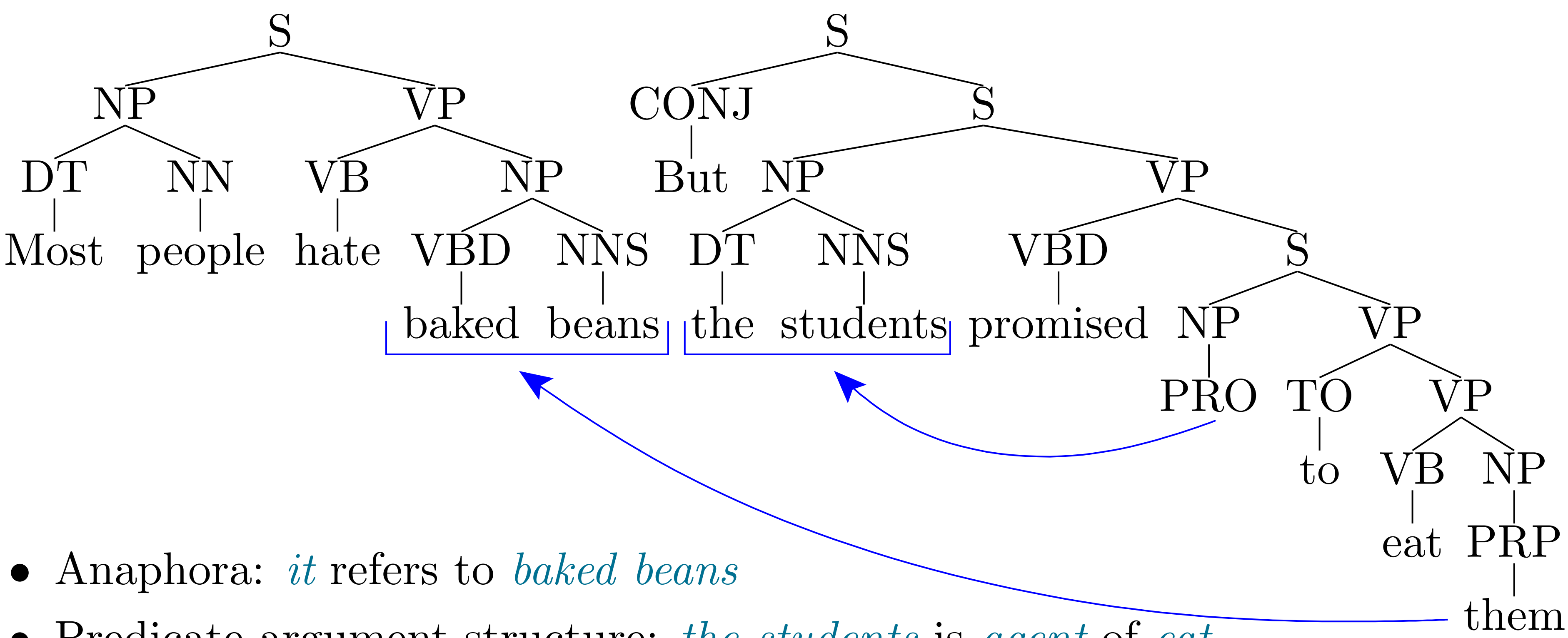
- \+ Language is a product of the human mind
  $\Rightarrow$ any structure we observe is a product of the mind
- – Language involves a *transduction between form and meaning*, but we don't know much about the way meanings are represented
- +/– We have (reasonable?) guesses about some of the computational processes involved in language
- – We don't know very much about the cognitive processes that language interacts with
- – We know little about the anatomical layout of language in the brain
- – We know little about neural networks that might support linguistic computations

Aspects of linguistic structure

- Phonetics: the (production and perception) of speech sounds
- Phonology: the organization and regularities of speech sounds
- Morphology: the structure and organization of words
- Syntax: the way words combine to form phrases and sentences
- Semantics: the way meaning is associated with sentences
- Pragmatics: how language can be used to do things

In general the further we get from speech, the less well we understand what's going on!

Aspects of syntactic and semantic structure



- Anaphora: *it* refers to *baked beans*
- Predicate-argument structure: *the students* is *agent* of *eat*
- Discourse structure: second clause is *contrasted* with first

These all refer to *phrase structure* entities! Parsing is the process of recovering these entities.

A very brief history

(Antiquity) Birth of linguistics, logic, rhetoric

(1900s) Structuralist linguistics (phrase structure)

(1900s) Mathematical logic

(1900s) *Probability and statistics*

(1940s) Behaviorism (discovery procedures, corpus linguistics)

(1940s) Ciphers and codes

(1950s) Information theory

(1950s) *Automata theory*

(1960s) *Context-free grammars*

(1960s) Generative grammar dominates (US) linguistics (Chomsky)

(1980s) “Neural networks” (learning as parameter estimation)

(1980s) *Graphical models* (Bayes nets, Markov Random Fields)

(1980s) Statistical models dominate speech recognition

(1980s) *Probabilistic grammars*

(1990s) Statistical methods dominate computational linguistics

(1990s) Computational learning theory

Topics

- *Graphical models and Bayes networks*
- (Hidden) Markov models
- (Probabilistic) context-free grammars
- (Probabilistic) finite-state machines
- Computation with PCFGs
- Estimation of PCFGs
- Lexicalized and bi-lexicalized PCFGs
- Non-local dependencies and log-linear models
- Stochastic unification-based grammars

Probability distributions

- A *probability distribution* over a countable set Ω is a function $\mathrm{P} : \Omega \to [0,1]$ which satisfies $1 = \sum_{\omega \in \Omega} \mathrm{P}(\omega)$.
- A *random variable* is a function $X : \Omega \to \mathcal{X}$. $\mathrm{P}(X{=}x) = \displaystyle\sum_{\omega : X(\omega) = x} \mathrm{P}(\omega)$
- If there are several random variables $X_1, \ldots, X_n$, then:
  - $\mathrm{P}(X_1, \ldots, X_n)$ is the *joint distribution*
  - $\mathrm{P}(X_i)$ is the *marginal distribution* of X_i
- $X_1, \ldots, X_n$ are *independent* iff $\mathrm{P}(X_1, \ldots, X_n) = \mathrm{P}(X_1) \ldots \mathrm{P}(X_n)$, i.e., the joint is the product of the marginals
- The *conditional distribution* of X given Y is $\mathrm{P}(X|Y) = \mathrm{P}(X, Y)/\mathrm{P}(Y)$ so $\mathrm{P}(X, Y) = \mathrm{P}(Y)\mathrm{P}(X|Y) = \mathrm{P}(X)\mathrm{P}(Y|X)$ *(Bayes rule)*
- $X_1, \ldots, X_n$ are *conditionally independent* given Y iff $\mathrm{P}(X_1, \ldots, X_n|Y) = \mathrm{P}(X_1|Y) \ldots \mathrm{P}(X_n|Y)$

Bayes inversion and the noisy channel model

Given an acoustic signal a, find words $\widehat{w}(a)$ most likely to correspond to a

$$w^\star(a) = \arg\max_w \mathrm{P}(W = w | A = a)$$

$$\mathrm{P}(A)\mathrm{P}(W|A) = \mathrm{P}(W, A) = \mathrm{P}(W)\mathrm{P}(A|W)$$

$$\mathrm{P}(W|A) = \frac{\mathrm{P}(W)\mathrm{P}(A|W)}{\mathrm{P}(A)}$$

$$\begin{aligned} w^\star(a) &= \arg\max_w \frac{\mathrm{P}(W = w)\mathrm{P}(A = a|W = w)}{\mathrm{P}(A = a)} \\ &= \arg\max_w \mathrm{P}(W = w)\mathrm{P}(A = a|W = w) \end{aligned}$$

$$\boxed{\begin{array}{c}\text{Language model}\\ \mathrm{P}(W)\end{array}} \;\downarrow\; \boxed{\begin{array}{c}\text{Acoustic model}\\ \mathrm{P}(A|W)\end{array}} \;\downarrow\; \text{Acoustic signal } A$$

Advantages of noisy channel model:

- $\mathrm{P}(W|A)$ is hard to construct directly; $\mathrm{P}(A|W)$ is easier
- noisy channel also exploits *language model* $\mathrm{P}(W)$

Why graphical models?

- Graphical models depict *factorizations of probability distributions*
- Statistical and computational properties depend on the factorization
  - complexity of dynamic programming is size of a certain cut in the graphical model
- Two different (but related) graphical representations
  - *Bayes nets* (directed graphs; products of conditionals)
  - *Markov Random Fields* (undirected graphs; products of arbitrary terms)
- Each random variable X_i is represented by a node

Bayes nets (directed graph)

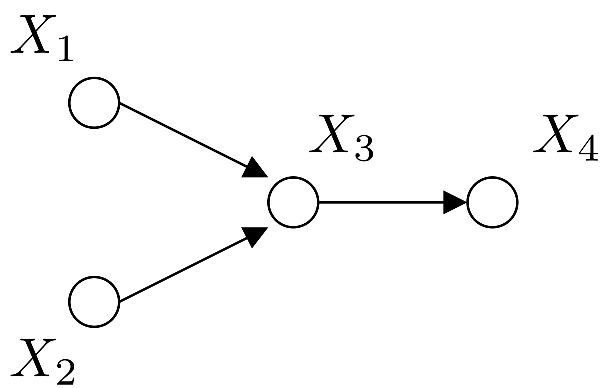
- Factorize *joint* $\mathrm{P}(X_1, \ldots, X_n)$ into *product of conditionals*

$$\mathrm{P}(X_1, \ldots, X_n) \quad = \quad \prod_{i=1}^{n} \mathrm{P}(X_i | X_{Pa(i)})$$

  where $Pa(i) \subseteq (X_1, \ldots, X_{i-1})$

- The *Bayes net* contains an arc from each $j \in Pa(i)$ to i

$$\mathrm{P}(X_1, X_2, X_3, X_4) \quad = \quad \mathrm{P}(X_1)\mathrm{P}(X_2)\mathrm{P}(X_3|X_1, X_2)\mathrm{P}(X_4|X_3)$$



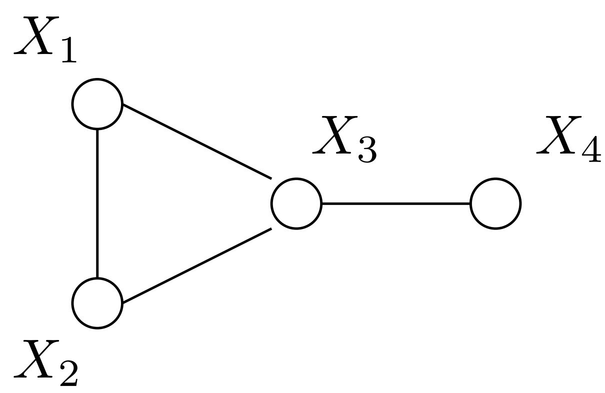
Markov Random Field (undirected)

- Factorize $\mathrm{P}(X_1, \ldots, X_n)$ into product of *potentials* $g_c(X_c)$, where $c \subseteq (1, \ldots, n)$ and $c \in \mathcal{C}$ (a set of tuples of indices)

$$\mathrm{P}(X_1, \ldots, X_n) = \frac{1}{Z} \prod_{c \in \mathcal{C}} g_c(X_c)$$

- If $i, j \in c \in \mathcal{C}$, then an edge connects i and j

$$\begin{aligned} \mathcal{C} &= \{(1,2,3),(3,4)\} \\ \mathrm{P}(X_1, X_2, X_3, X_4) &= \frac{1}{Z}\, g_{123}(X_1, X_2, X_3)\, g_{34}(X_3, X_4) \end{aligned}$$



A rose by any other name ...

- MRFs have the same form as *Maximum Entropy models*, *Exponential models*, *Log-linear models*, *Harmony models*, ...

$$
\begin{aligned}
\mathrm{P}(X) &= \frac{1}{Z} \prod_{c \in \mathcal{C}} g_c(X_c) \\
&= \frac{1}{Z} \prod_{c \in \mathcal{C}, x_c \in \mathcal{X}_c} (\theta_{X_c = x_c})^{[\![X_c = x_c]\!]}, \text{ where } \theta_{X_c = x_c} = g_c(x_c) \\
&= \frac{1}{Z} \exp \sum_{c \in \mathcal{C}, X_c \in \mathcal{X}_c} [\![X_c = x_c]\!] \phi_{X_c = x_c}, \text{ where } \phi_{X_c = x_c} = \log g_c(x_c)
\end{aligned}
$$

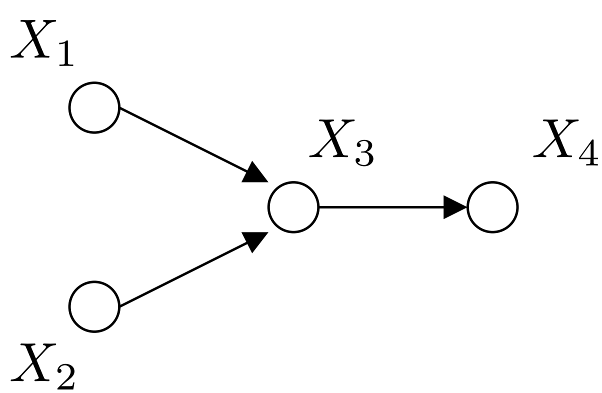
$$
\begin{aligned}
\mathrm{P}(X) &= \frac{1}{Z} \, g_{123}(X_1, X_2, X_3) \; g_{34}(X_3, X_4) \\
&= \frac{1}{Z} \exp \left(\begin{array}{c} [\![X_{123} = 000]\!] \phi_{000} + [\![X_{123} = 001]\!] \phi_{001} + \ldots \\ [\![X_{34} = 00]\!] \phi_{00} + [\![X_{34} = 01]\!] \phi_{01} + \ldots \end{array} \right)
\end{aligned}
$$

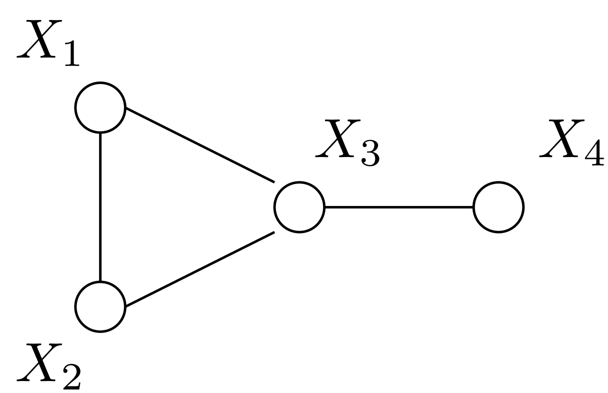
Bayes nets and MRFs

- MRFs are more general than Bayes nets
- Its easy to find the MRF representation of a Bayes net

$$\mathrm{P}(X_1, X_2, X_3, X_4) = \underbrace{\mathrm{P}(X_1)\mathrm{P}(X_2)\mathrm{P}(X_3|X_1, X_2)}_{g_{123}(X_1, X_2, X_3)} \underbrace{\mathrm{P}(X_4|X_3)}_{g_{34}(X_3, X_4)}$$

- *Moralization*, i.e, "marry the parents"

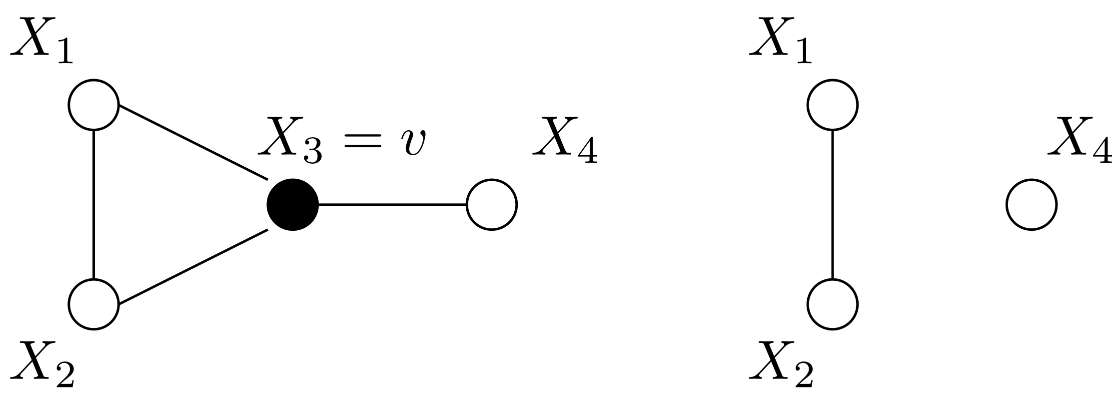




Conditionalization in MRFs

- Conditionalization is *fixing the value of certain variables*
- To get a MRF representation of the conditional distribution, *delete nodes whose values are fixed and arcs connected to them*

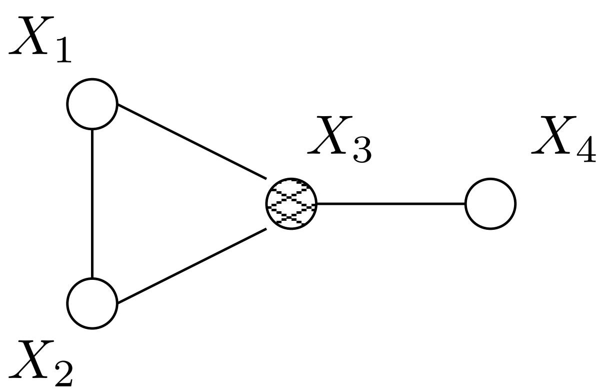
$$\begin{aligned} \mathrm{P}(X_1, X_2, X_4 | X_3 = v) &= \frac{1}{Z \;\; \mathrm{P}(X_3 = v)} \, g_{123}(X_1, X_2, v) \; g_{34}(v, X_4) \\ &= \frac{1}{Z'(v)} \quad g'_{12}(X_1, X_2) \quad g'_4(X_4) \end{aligned}$$

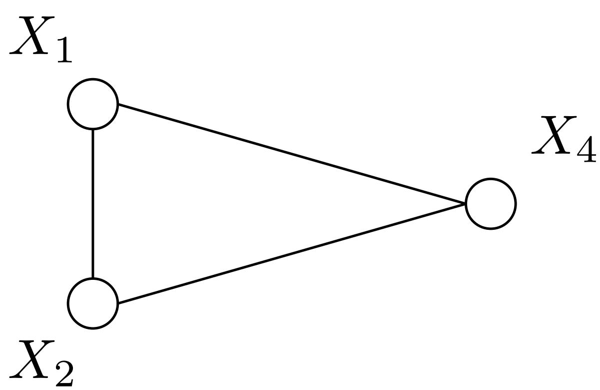


Marginalization in MRFs

- Marginalization is *summing over all possible values of certain variables*
- To get a MRF representation of the marginal distribution, *delete the marginalized nodes and interconnect all of their neighbours*

$$
\begin{aligned}
\mathrm{P}(X_1, X_2, X_4) &= \sum_{X_3} \mathrm{P}(X_1, X_2, X_3, X_4) \\
&= \sum_{X_3} \; g_{123}(X_1, X_2, X_3)\; g_{34}(X_3, X_4) \\
&= g'_{124}(X_1, X_2, X_4)
\end{aligned}
$$





Classification

- Given value of X, predict value of Y
- Given a probabilistic model $\mathrm{P}(Y|X)$, predict:

$$y^\star(x) \quad = \quad \arg\max_y \mathrm{P}(y|x)$$

- Learn $\mathrm{P}(Y|X)$ from data $D = ((x_1, y_1), \ldots, (x_n, y_n))$
- Restrict attention to a *parametric model class* P_θ parameterized by parameter vector θ
  - learning is estimating θ from D

ML and CML Estimation

- Maximum likelihood estimation (MLE) picks the θ that makes the data $D = (x, y)$ as *likely as possible*

$$\widehat{\theta} \quad = \quad \arg\max_{\theta} \mathrm{P}_{\theta}(x, y)$$

- Conditional maximum likelihood estimation (CMLE) picks the θ that maximizes *conditional likelihood* of the data $D = (x, y)$

$$\widehat{\theta}' \quad = \quad \arg\max_{\theta} \mathrm{P}_{\theta}(y|x)$$

- $\mathrm{P}(X, Y) = \mathrm{P}(X)\mathrm{P}(Y|X)$, so CMLE *ignores* $\mathrm{P}(X)$

MLE and CMLE example

- $X, Y \in \{0,1\}$, $\theta \in [0,1]$, $\mathrm{P}_\theta(X=1)=\theta$, $\mathrm{P}_\theta(Y=X|X)=\theta$

  Choose X by flipping a coin with weight θ, then set Y to same value as X if flipping same coin again comes out 1.

- Given data $D = ((x_1, y_1), \ldots, (x_n, y_n))$,

$$\begin{aligned}\widehat{\theta} &= \frac{\sum_i^n [\![x_i = 1]\!] + [\![x_i = y_i]\!]}{2n} \\ \widehat{\theta}' &= \frac{\sum_i^n [\![x_i = y_i]\!]}{n}\end{aligned}$$

- CMLE *ignores* $\mathrm{P}(X)$, so *less efficient* if model *correctly* relates $\mathrm{P}(Y|X)$ and $\mathrm{P}(X)$

- But if model *incorrectly relates* $\mathrm{P}(Y|X)$ and $\mathrm{P}(X)$, MLE converges to wrong θ
  - e.g., if x_i are chosen by some different process entirely

Complexity of decoding and estimation

- Finding $y^\star(x) = \arg\max_y \mathrm{P}(y|x)$ is *equally hard* for Bayes nets and MRFs with similar architectures

- A Bayes net is a product of independent conditional probabilities

  $\Rightarrow$ MLE is *relative frequency* (easy to compute)

  – *no closed form for CMLE* if conditioning variables have parents

- A MRF is a product of arbitrary potential functions g

  – estimation involves learning values of each g takes

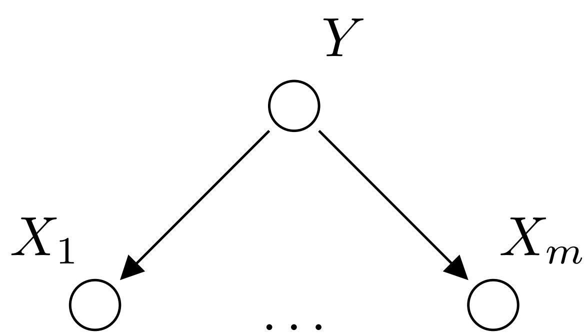
  – partition function Z changes as we adjust g

  $\Rightarrow$ usually *no closed form for MLE and CMLE*

Multiple features and Naive Bayes

- Predict label Y from features $X_1, \ldots, X_m$

$$
\begin{aligned}
\mathrm{P}(Y|X_1,\ldots,X_m) &\propto \mathrm{P}(Y)\prod_{j=1}^{m}\mathrm{P}(X_j|Y,X_1,\ldots,X_{j-1}) \\
&\approx \mathrm{P}(Y)\prod_{j=1}^{m}\mathrm{P}(X_j|Y)
\end{aligned}
$$

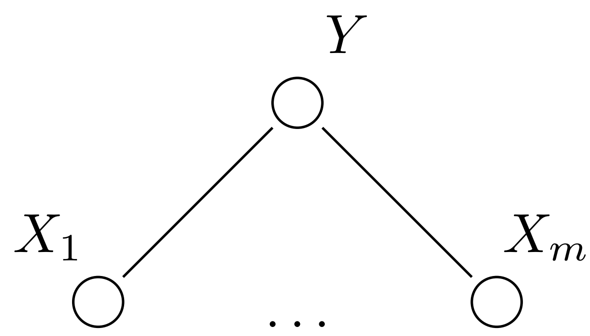


- Naive Bayes estimate is MLE $\widehat{\theta} = \arg\max_\theta \mathrm{P}(x_1, \ldots, x_n, y)$
  - Trivial to compute (relative frequency)
  - May be poor if X_j aren't really conditionally independent

Multiple features and MaxEnt

- Predict label Y from features $X_1, \ldots, X_m$

$$\mathrm{P}(Y|X_1, \ldots, X_m) \quad \propto \quad \prod_{j=1}^{m} g_j(X_j, Y)$$

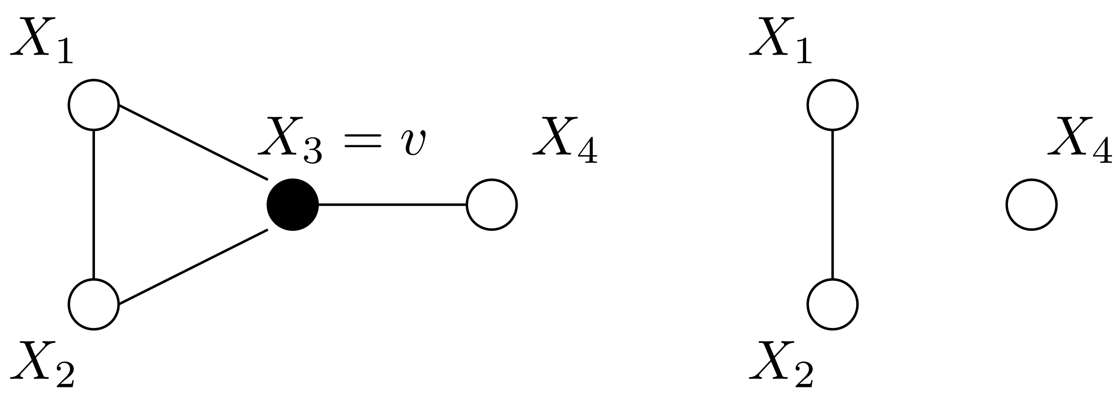


- MaxEnt estimate is CMLE $\widehat{\theta}' = \arg\max_\theta \mathrm{P}(y|x_1, \ldots, x_m)$
  - Makes *no assumptions* about $\mathrm{P}(X)$
  - Difficult to compute (iterative numerical optimization)

Conditionalization in MRFs

- Conditionalization is *fixing the value of certain variables*
- To get a MRF representation of the conditional distribution, *delete nodes whose values are fixed and arcs connected to them*

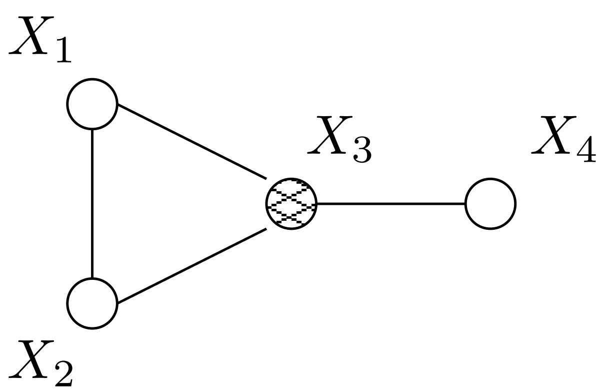
$$
\begin{aligned}
\mathrm{P}(X_1, X_2, X_4 | X_3 = v) &= \frac{1}{Z \ \mathrm{P}(X_3 = v)} \, g_{123}(X_1, X_2, v) \; g_{34}(v, X_4) \\
&= \frac{1}{Z'(v)} \quad g'_{12}(X_1, X_2) \qquad g'_4(X_4)
\end{aligned}
$$

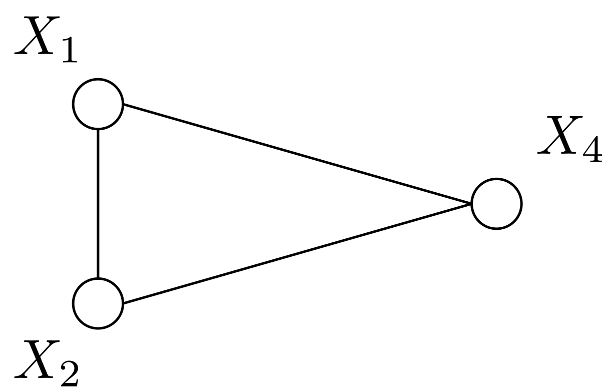


Marginalization in MRFs

- Marginalization is *summing over all possible values of certain variables*
- To get a MRF representation of the marginal distribution, *delete the marginalized nodes and interconnect all of their neighbours*

$$
\begin{aligned}
\mathrm{P}(X_1, X_2, X_4) &= \sum_{X_3} \mathrm{P}(X_1, X_2, X_3, X_4) \\
&= \sum_{X_3} g_{123}(X_1, X_2, X_3)\, g_{34}(X_3, X_4) \\
&= g'_{124}(X_1, X_2, X_4)
\end{aligned}
$$





Computation in MRFs

- Given a MRF describing a probability distribution

$$\mathrm{P}(X_1, \ldots, X_n) \quad = \quad \frac{1}{Z} \prod_{c \in \mathcal{C}} g_c(X_c)$$

  where each X_c is a subset of $X_1, \ldots, X_n$, involve sum/max of products expressions

$$\begin{aligned}
Z \quad &= \quad \sum_{X_1, \ldots, X_n} \prod_{c \in \mathcal{C}} g_c(X_c) \\
\mathrm{P}(X_i = x_i) \quad &= \quad \frac{1}{Z} \sum_{X_1, \ldots, X_{i-1}, X_{i+1}, X_n} \prod_{c \in \mathcal{C}} g_c(X_c) \text{ with } X_i = x_i \\
x_i^\star \quad &= \quad \arg\max_{X_i} \sum_{X_1, \ldots, X_{i-1}, X_{i+1}, X_n} \prod_{c \in \mathcal{C}} g_c(X_c)
\end{aligned}$$

- Dynamic programming involves *factorizing* the sum/max of products expression

Factorizing a sum/max of products

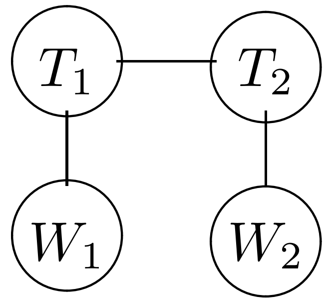
Order the variables, *repeatedly marginalize* each variable, and introduce a new auxiliary function c_i for each marginalized variable X_i.

$$
\begin{aligned}
Z &= \sum_{X_1,\ldots,X_n} \prod_{c\in\mathcal{C}} g_c(X_c) \\
&= \sum_{X_n}(\ldots(\sum_{X_1}\ldots)\ldots)
\end{aligned}
$$

See Geman and Kochanek, 2000, “Dynamic Programming and the Representation of Soft-Decodable Codes”

MRF factorization example (1)

W_1, W_2 are adjacent words, and T_1, T_2 are their POS.



$$
\begin{aligned}
\mathrm{P}(W_1, W_2, T_1, T_2) &= \frac{1}{Z} g(W_1, T_1) h(T_1, T_2) g(W_2, T_2) \\
Z &= \sum_{W_1, T_1, W_2, T_2} g(W_1, T_1) h(T_1, T_2) g(W_2, T_2)
\end{aligned}
$$

$|\mathcal{W}|^2 |\mathcal{T}|^2$ different combinations of variable values in direct enumeration of Z

MRF factorization example (2)

$$
\begin{aligned}
Z &= \sum_{W_1,T_1,W_2,T_2} g(W_1,T_1)h(T_1,T_2)g(W_2,T_2) \\
&= \sum_{T_1,W_2,T_2} (\sum_{W_1} g(W_1,T_1))h(T_1,T_2)g(W_2,T_2) \\
&= \sum_{T_1,W_2,T_2} c_{W_1}(T_1)h(T_1,T_2)g(W_2,T_2) \text{ where } c_{W_1}(T_1) = \textstyle\sum_{W_1} g(W_1,T_1) \\
&= \sum_{W_2,T_2} (\sum_{T_1} c_{W_1}(T_1)h(T_1,T_2))g(W_2,T_2) \\
&= \sum_{W_2,T_2} c_{T_1}(T_2)g(W_2,T_2) \text{ where } c_{T_1}(T_2) = \textstyle\sum_{T_1} c_{W_1}(T_1)h(T_1,T_2) \\
&= \sum_{W_2}(\sum_{T_2} c_{T_1}(T_2)g(W_2,T_2)) \\
&= \sum_{W_2} c_{T_2}(W_2) \text{ where } c_{T_2}(W_2) = \textstyle\sum_{T_2} c_{T_1}(T_2)g(W_2,T_2) \\
&= c_{W_2} \text{ where } c_{W_2} = \textstyle\sum_{W_2} c_{T_2}(W_2)
\end{aligned}
$$

MRF factorization example (3)

$$
\begin{aligned}
Z &= c_{W_2} \\
c_{W_2} &= \sum_{W_2} c_{T_2}(W_2) && (|\mathcal{W}|\text{operations}) \\
c_{T_2}(W_2) &= \sum_{T_2} c_{T_1}(T_2) g(W_2, T_2) && (|\mathcal{W}||\mathcal{T}|\text{operations}) \\
c_{T_1}(T_2) &= \sum_{T_1} c_{W_1}(T_1) h(T_1, T_2) && (|\mathcal{T}|^2\text{operations}) \\
c_{W_1}(T_1) &= \sum_{W_1} g(W_1, T_1) && (|\mathcal{W}||\mathcal{T}|\text{operations})
\end{aligned}
$$

So computing Z in this way $|\mathcal{W}| + 2|\mathcal{W}||\mathcal{T}| + |\mathcal{T}|^2$ operations, as opposed to $|\mathcal{W}|^2|\mathcal{T}|^2$ operations for direct enumeration

Factoring sum/max product expressions

- In general the function c_j for marginalizing X_j will have X_k as an argument if there is an arc from X_i to X_k for some $i \leq j$
- Computational complexity is exponential in the number of arguments to these functions c_j
- Finding the optimal ordering of variables that minimizes computational complexity for arbitrary graphs is NP-hard

Topics

- Graphical models and Bayes networks
- *Markov chains and hidden Markov models*
- (Probabilistic) context-free grammars
- (Probabilistic) finite-state machines
- Computation with PCFGs
- Estimation of PCFGs
- Lexicalized and bi-lexicalized PCFGs
- Non-local dependencies and log-linear models
- Stochastic unification-based grammars

Markov chains

Let $X = X_1, \ldots, X_n, \ldots$, where each $X_i \in \mathcal{X}$.

By Bayes rule: $\mathrm{P}(X_1, \ldots, X_n) \;=\; \prod_{i=1}^{n} \mathrm{P}(X_i | X_1, \ldots, X_{i-1})$

X is a *Markov chain* iff $\mathrm{P}(X_i | X_1, \ldots, X_{i-1}) = \mathrm{P}(X_i | X_{i-1})$, i.e.,

$$\mathrm{P}(X_1, \ldots, X_n) \;=\; P(X_1) \prod_{i=2}^{n} \mathrm{P}(X_i | X_{i-1})$$

Bayes net representation of a Markov chain:

$$X_1 \longrightarrow X_2 \longrightarrow \ldots \longrightarrow X_{i-1} \longrightarrow X_i \longrightarrow X_{i+1} \longrightarrow \ldots$$

A Markov chain is *homogeneous* or *time-invariant* iff
$\mathrm{P}(X_i | X_{i-1}) = \mathrm{P}(X_j | X_{j-1})$ for all i, j

A homogeneous Markov chain is completely specified by

- *start probabilities* $p_s(x) = \mathrm{P}(X_1 = x)$, and
- *transition probabilities* $p_m(x | x') = \mathrm{P}(X_i = x | X_{i-1} = x')$

Bigram models

A *bigram language model* B defines a probability distribution over strings of words $w_1 \dots w_n$ based on the word pairs (w_i, w_{i+1}) the string contains.

A bigram model is a homogenous Markov chain:

$$\mathrm{P}_B(w_1 \dots w_n) \;=\; p_s(w_1) \prod_{i=1}^{n-1} p_m(w_{i+1}|w_i)$$

$$W_1 \longrightarrow W_2 \longrightarrow \dots \longrightarrow W_{i-1} \longrightarrow W_i \longrightarrow W_{i+1} \longrightarrow \dots$$

We need to define a distribution over the *lengths* n of strings. One way to do this is by appending an *end-marker* \$ to each string, and set $p_m(\$|\$) = 1$

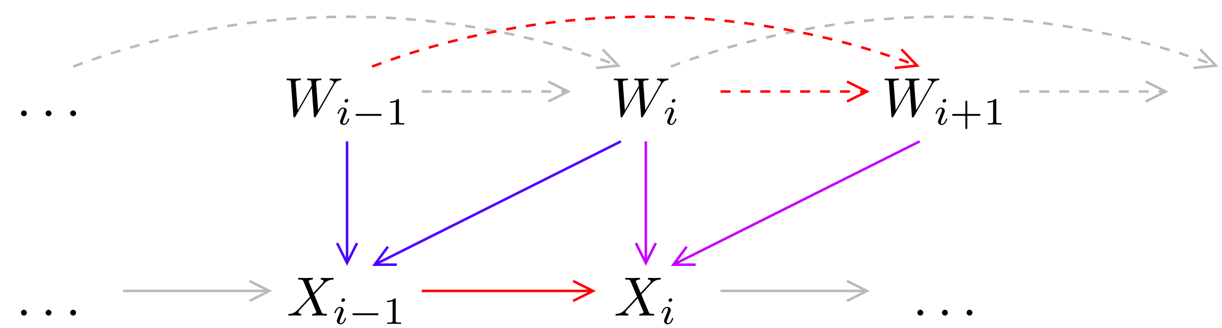
$\mathrm{P}(\text{Howard hates brocolli \$})$
$= \; p_s(\text{Howard})p_m(\text{hates}|\text{Howard})p_m(\text{brocolli}|\text{hates})p_m(\$|\text{brocolli})$

n-gram models

An *m-gram model* L_n defines a probability distribution over strings based on the m-tuples $(w_i, \ldots, w_{i+m-1})$ the string contains.

An m-gram model is also a homogenous Markov chain, where the chain's random variables are $m-1$ *tuples* of words $X_i = (W_i, \ldots, W_{i+m-2})$. Then:

$$\begin{aligned} \mathrm{P}_{L_n}(W_1, \ldots, W_{n+m-2}) &= \mathrm{P}_{L_n}(X_1 \ldots X_n) = p_s(x_1) \prod_{i=1}^{n-1} p_m(x_{i+1}|x_i) \\ &= p_s(w_1, \ldots, w_{m-1}) \prod_{j=m}^{n+m-2} p_m(w_j|w_{j-1}, \ldots, w_{j-m+1}) \end{aligned}$$



$$\mathrm{P}_{L_3}(\text{Howard likes brocolli \$}) = p_s(\text{Howard likes}) p_m(\text{brocolli}|\text{Howard likes}) p_m(\$|\text{likes brocolli})$$

Sequence labeling

- Predict hidden labels $S_1, \ldots, S_m$ given visible features $V_1, \ldots, V_m$
- Example: Parts of speech

$$\begin{array}{lcccccc} S & = & \text{DT} & \text{JJ} & \text{NN} & \text{VBS} & \text{JJR} \\ V & = & \text{the} & \text{big} & \text{dog} & \text{barks} & \text{loudly} \end{array}$$

- Example: Named entities

$$\begin{array}{lcccccc} S & = & \text{[NP} & \text{NP} & \text{NP]} & - & - \\ V & = & \text{the} & \text{big} & \text{dog} & \text{barks} & \text{loudly} \end{array}$$

Hidden Markov models

A *hidden variable* is one whose value cannot be directly observed.

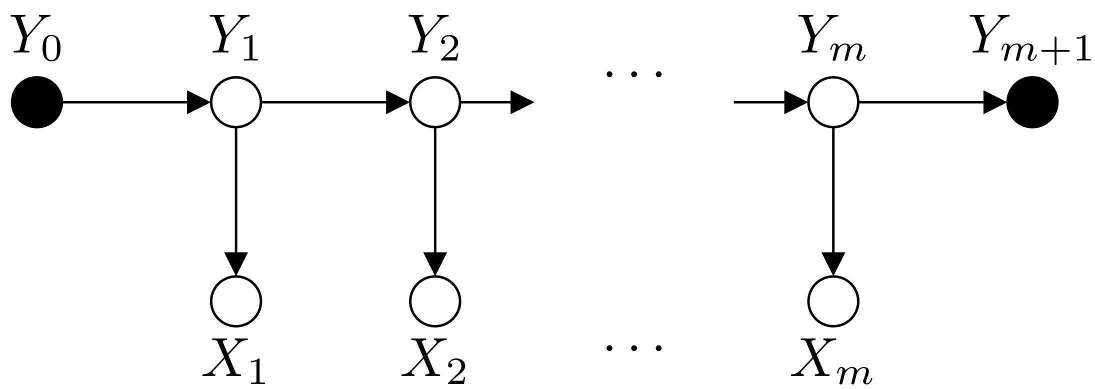
In a *hidden Markov model* the *state sequence* $S_1 \ldots S_n \ldots$ is a hidden Markov chain, but each state S_i is associated with a visible output V_i.

$$\mathrm{P}(S_1, \ldots, S_n; V_1, \ldots, V_n) \;=\; \mathrm{P}(S_1)\mathrm{P}(V_1|S_1) \prod_{i=1}^{n-1} \mathrm{P}(S_{i+1}|S_i)\mathrm{P}(V_{i+1}|S_{i+1})$$

$$\begin{tikzcd}
\ldots \arrow[r] & S_{i-1} \arrow[r] \arrow[d] & S_i \arrow[r] \arrow[d] & S_{i+1} \arrow[r] \arrow[d] & \ldots \\
& V_{i-1} & V_i & V_{i+1} &
\end{tikzcd}$$

Hidden Markov Models

$$\mathrm{P}(X, Y) \quad = \quad \left(\prod_{j=1}^{m} \mathrm{P}(Y_j | Y_{j-1}) \mathrm{P}(X_j | Y_j) \right) \mathrm{P}(Y_m, stop)$$



- Usually assume *time invariance* or *stationarity*
  i.e., $\mathrm{P}(Y_j | Y_{j-1})$ and $\mathrm{P}(X_j | Y_j)$ do not depend on j

- *HMMs are Naive Bayes models with compound labels Y*

- Estimator is MLE $\widehat{\theta} = \arg\max_{\theta} \mathrm{P}_{\theta}(x, y)$

Applications of homogeneous HMMs

Acoustic model in speech recognition: $\mathrm{P}(A|W)$

States are *phonemes*, outputs are *acoustic features*

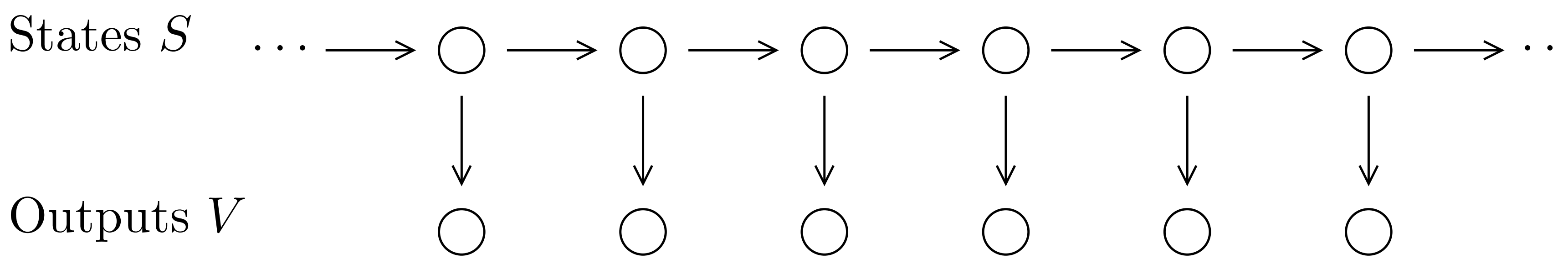
$$\begin{tikzcd}
\ldots \arrow[r] & S_{i-1} \arrow[r] \arrow[d] & S_i \arrow[r] \arrow[d] & S_{i+1} \arrow[r] \arrow[d] & \ldots \\
& V_{i-1} & V_i & V_{i+1} &
\end{tikzcd}$$

Part of speech tagging:

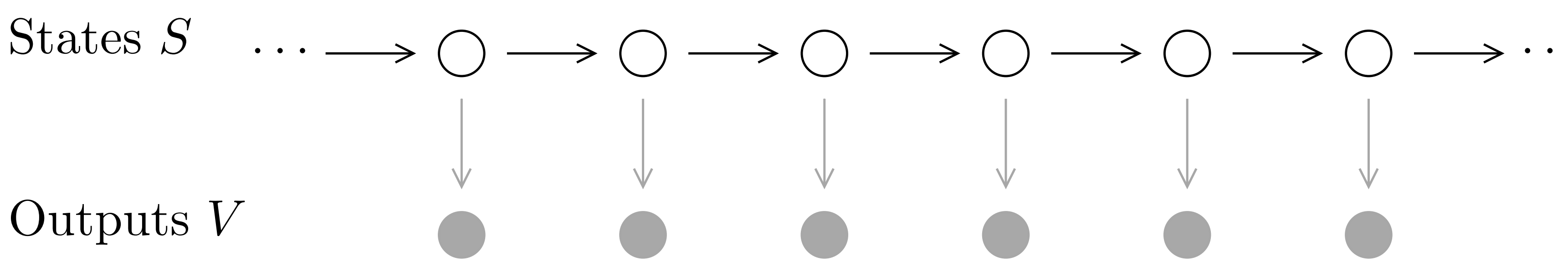
States are *parts of speech*, outputs are *words*

$$\begin{tikzcd}
\mathsf{NNP} \arrow[r] \arrow[d] & \mathsf{VB} \arrow[r] \arrow[d] & \mathsf{NNS} \arrow[r] \arrow[d] & \$ \arrow[d] \\
\mathsf{Howard} & \mathsf{likes} & \mathsf{mangoes} & \$
\end{tikzcd}$$

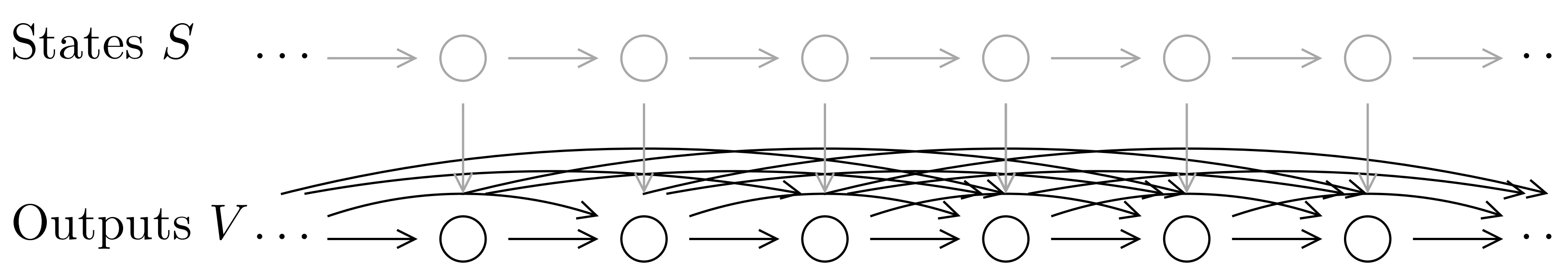
Properties of HMMs



Conditioning on outputs $\mathrm{P}(S|V)$ results in *Markov state dependencies*

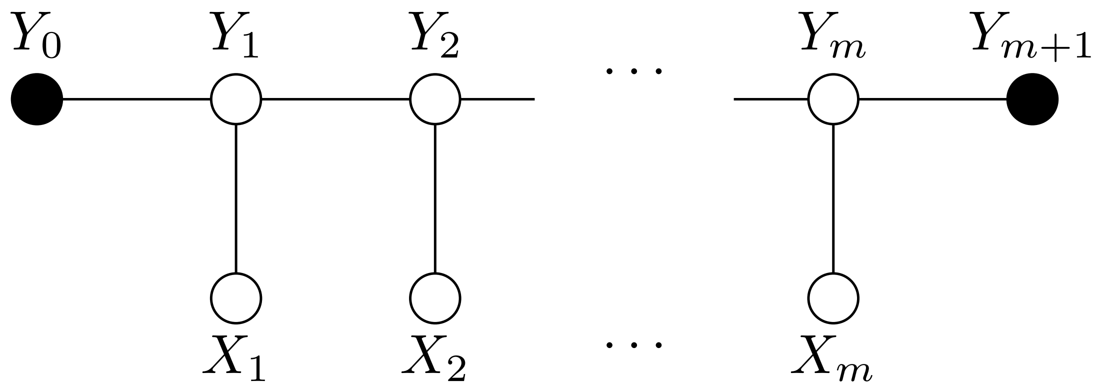


Marginalizing over states $\mathrm{P}(V) = \sum_S \mathrm{P}(S, V)$ *completely connects outputs*



Conditional Random Fields

$$\mathrm{P}(Y|X) \quad = \quad \frac{1}{Z(x)}\left(\prod_{j=1}^{m} f(Y_j, Y_{j-1})g(X_j, Y_j)\right) f(Y_m, stop)$$



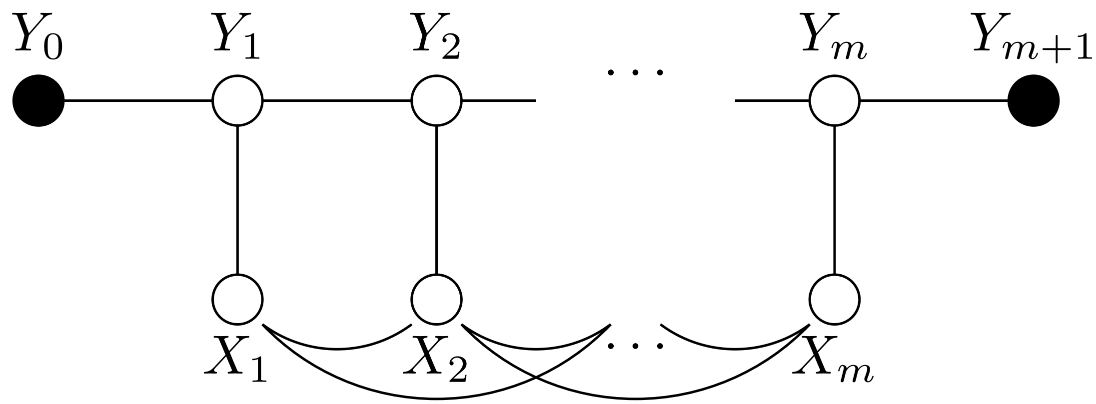
- *time invariance* or *stationarity*, i.e., f and g don't depend on j
- CRFs are MaxEnt models with compound labels Y
- Estimator is CMLE $\widehat{\theta}' = \arg\max_\theta \mathrm{P}_\theta(y|x)$

Decoding and Estimation

- HMMs and CRFs have *same complexity of decoding* i.e., computing $y^\star(x) = \arg\max_y \mathrm{P}(y|x)$
  - dynamic programming algorithm (Viterbi algorithm)
- Estimating a HMM from labeled data (x, y) is *trivial*
  - HMMs are Bayes nets $\Rightarrow$MLE is relative frequency
- Estimating a CRF from labeled data (x, y) is *difficult*
  - Usually *no closed form* for partition function $Z(x)$
  - Use *iterative numerical optimization procedures* (e.g., Conjugate Gradient, Limited Memory Variable Metric) to maximize $\mathrm{P}_\theta(y|x)$

When are CRFs better than HMMs?

- When HMM independence assumptions are wrong, i.e., there are dependences between X_j not described in model

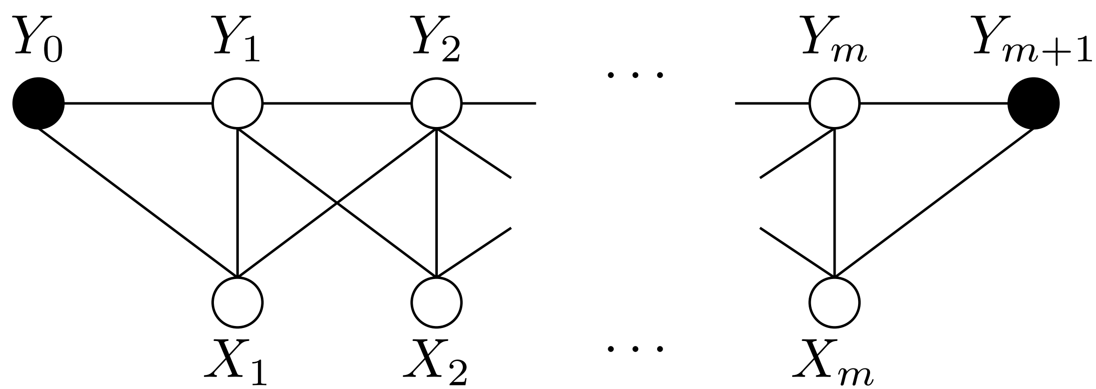


- HMM uses MLE $\Rightarrow$models joint $\mathrm{P}(X, Y) = \mathrm{P}(X)\mathrm{P}(Y|X)$
- CRF uses CMLE $\Rightarrow$models conditional distribution $\mathrm{P}(Y|X)$
- Because CRF uses CMLE, it makes no assumptions about $\mathrm{P}(X)$
- *If* $\mathrm{P}(X)$ *isn't modeled well by HMM, don't use HMM!*

Overlapping features

- Sometimes label Y_j depends on X_{j-1} and X_{j+1} as well as X_j

$$\mathrm{P}(Y|X) \quad = \quad \frac{1}{Z(x)} \left(\prod_{j=1}^{m} f(X_j, Y_j, Y_{j-1}) g(X_j, Y_j, Y_{j+1}) \right)$$



- Most people think this would be difficult to do in a HMM

Summary

- HMMs and CRFs both associate a *sequence* of labels $(Y_1, \ldots, Y_m)$ to items $(X_1, \ldots, X_m)$
- HMMs are Bayes nets and estimated by MLE
- CRFs are MRFs and estimated by CMLE
- HMMs assume that X_j are *conditionally independent*
- CRFs do not assume that the X_j are conditionally independent
- The Viterbi algorithm computes $y^\star(x)$ for both HMMs and CRFs
- HMMs are trivial to estimate
- CRFs are difficult to estimate
- It is easier to add new features to a CRF
- There is no EM version of CRF

Topics

- Graphical models and Bayes networks
- Markov chains and hidden Markov models
- *(Probabilistic) context-free grammars*
- (Probabilistic) finite-state machines
- Computation with PCFGs
- Estimation of PCFGs
- Lexicalized and bi-lexicalized PCFGs
- Non-local dependencies and log-linear models
- Stochastic unification-based grammars

Languages and Grammars

If $\mathcal{V}$ is a set of symbols (the *vocabulary*, i.e., words, letters, phonemes, etc):

- $\mathcal{V}^\star$ is the set of *all strings* (or finite sequences) of members of $\mathcal{V}$ (including the *empty sequence* ϵ)
- $\mathcal{V}^+$ is the set of all finite non-empty strings of members of $\mathcal{V}$

A *language* is a subset of $\mathcal{V}^\star$ (i.e., a set of strings)

A *probabilistic language* is probability distribution P over $\mathcal{V}^\star$, i.e.,

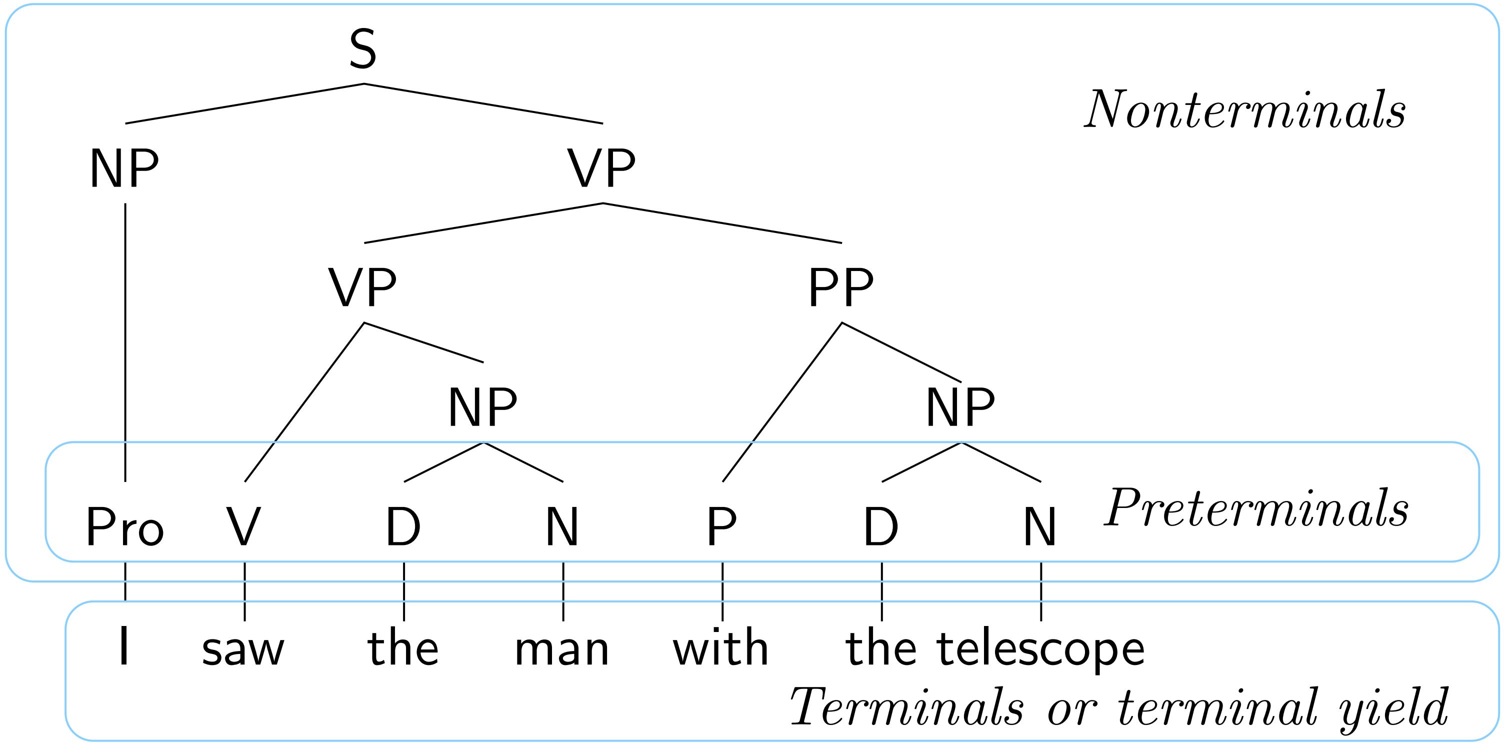
- $\forall w \in \mathcal{V}^\star \; 0 \leq \mathrm{P}(w) \leq 1$
- $\sum_{w \in \mathcal{V}^\star} \mathrm{P}(w) = 1$, i.e., P is *normalized*

A *(probabilistic) grammar* is a finite specification of a (probabilistic) language

Trees depict constituency

Some grammars G define a language by defining a set of trees Ψ_G.

The strings G generates are the *terminal yields* of these trees.



Trees represent how words combine to form phrases and ultimately sentences.

Probabilistic grammars

Some probabilistic grammars G defines a probability distribution $\mathrm{P}_G(\psi)$ over the set of trees Ψ_G, and hence over strings $w \in \mathcal{V}^\star$.

$$\mathrm{P}_G(w) \;=\; \sum_{\psi \in \Psi_G(w)} \mathrm{P}_G(\psi)$$

where $\Psi_G(w)$ are the trees with yield w generated by G

Standard (non-stochastic) grammars distinguish *grammatical* from *ungrammatical* strings (only the grammatical strings receive parses).

Probabilistic grammars can assign non-zero probability to every string, and rely on the probability distribution to distinguish likely from unlikely strings.

Context free grammars

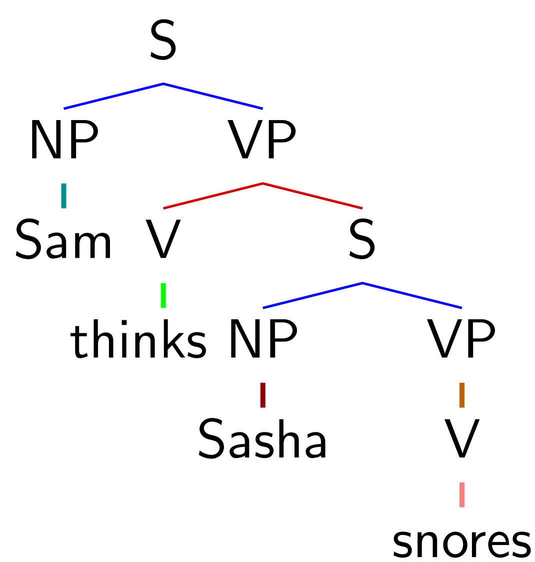
A *context-free grammar* $G = (\mathcal{V}, \mathcal{S}, s, \mathcal{R})$ consists of:

- $\mathcal{V}$, a finite set of *terminals* ($\mathcal{V}_0 = \{\mathsf{Sam}, \mathsf{Sasha}, \mathsf{thinks}, \mathsf{snores}\}$)
- $\mathcal{S}$, a finite set of *non-terminals* disjoint from $\mathcal{V}$ ($\mathcal{S}_0 = \{\mathsf{S}, \mathsf{NP}, \mathsf{VP}, \mathsf{V}\}$)
- $\mathcal{R}$, a finite set of *productions* of the form $A \to X_1 \dots X_n$, where $A \in \mathcal{S}$ and each $X_i \in \mathcal{S} \cup \mathcal{V}$
- $s \in \mathcal{S}$ is called the *start symbol* ($s_0 = \mathsf{S}$)

G *generates* a tree ψ iff

- The label of ψ's root node is s
- For all *local trees* with parent A and children $X_1 \dots X_n$ in ψ $A \to X_1 \dots X_n \in \mathcal{R}$

G generates a string $w \in \mathcal{V}^\star$ iff w is the *terminal yield* of a tree generated by G



Productions

S$\to$ NP VP

NP$\to$ Sam

NP$\to$ Sasha

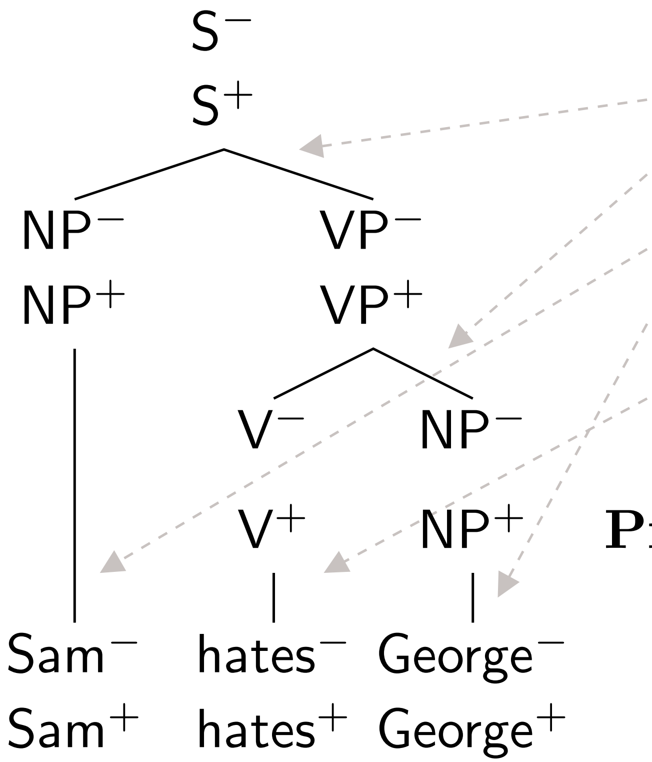
VP$\to$ V

VP$\to$ V S

V$\to$ thinks

V$\to$ snores

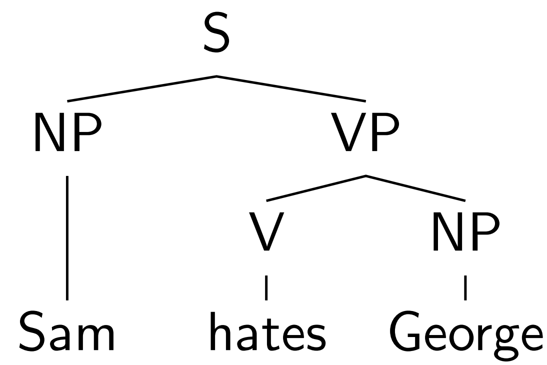
CFGs as “plugging” systems



“Pluggings”

S$\longrightarrow$ NP VP
VP$\longrightarrow$ V NP
NP$\longrightarrow$ Sam
NP$\longrightarrow$ George
V$\longrightarrow$ hates
V$\longrightarrow$ likes

Productions

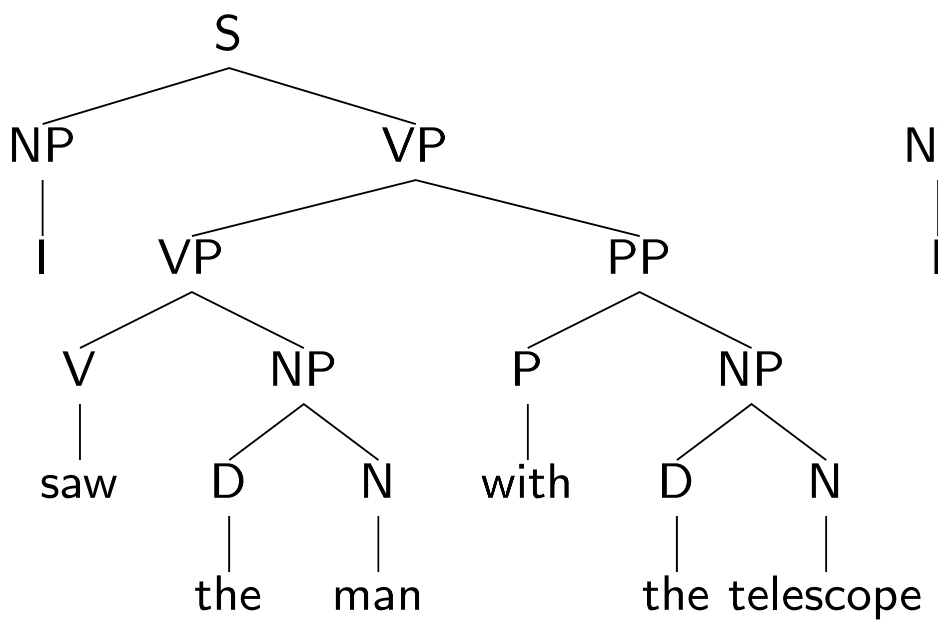


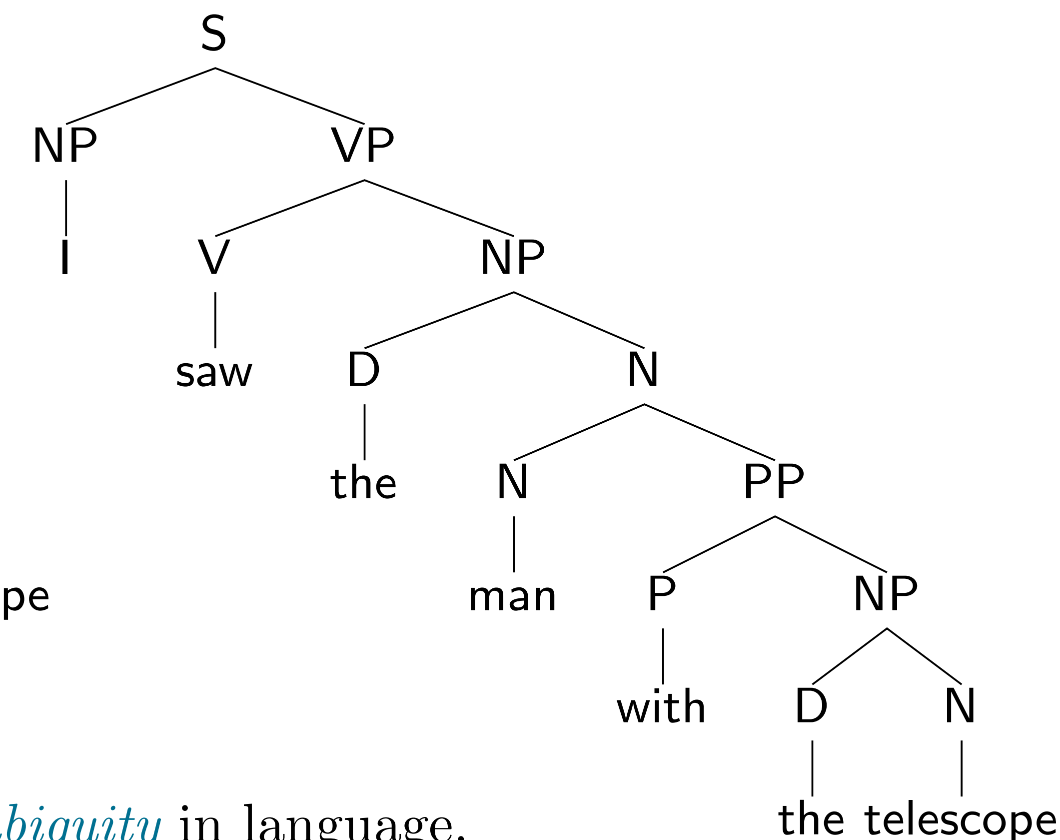
Resulting tree

- Goal: no unconnected “sockets” or “plugs”
- The *productions* specify available types of components
- In a *probabilistic* CFG each type of component has a “price”

Structural Ambiguity

$\mathcal{R}_1 = \{\mathsf{VP} \rightarrow \mathsf{V\ NP}, \mathsf{VP} \rightarrow \mathsf{VP\ PP}, \mathsf{NP} \rightarrow \mathsf{D\ N}, \mathsf{N} \rightarrow \mathsf{N\ PP}, \ldots\}$





- CFGs can capture *structural ambiguity* in language.
- Ambiguity generally grows *exponentially* in the length of the string.
  - The number of ways of parenthesizing a string of length n is Catalan(n)
- Broad-coverage statistical grammars are astronomically ambiguous.

Derivations

A CFG $G = (\mathcal{V}, \mathcal{S}, s, \mathcal{R})$ induces a *rewriting relation* $\Rightarrow_G$, where $\gamma A \delta \Rightarrow_G \gamma\beta\delta$ iff $A \rightarrow \beta \in \mathcal{R}$ and $\gamma, \delta \in (\mathcal{S} \cup \mathcal{V})^\star$.

A *derivation* of a string $w \in \mathcal{V}^\star$ is a finite sequence of rewritings $s \Rightarrow_G \ldots \Rightarrow_G w$. $\Rightarrow^\star_G$ is the *reflexive and transitive closure* of $\Rightarrow_G$.

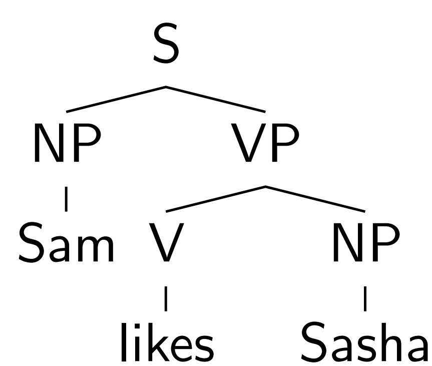
The *language generated* by G is $\{w : s \Rightarrow^\star w, w \in \mathcal{V}^\star\}$.

$G_0 = (\mathcal{V}_0, \mathcal{S}_0, \mathsf{S}, \mathcal{R}_0)$, $\mathcal{V}_0 = \{\mathsf{Sam}, \mathsf{Sasha}, \mathsf{likes}, \mathsf{hates}\}$, $\mathcal{S}_0 = \{\mathsf{S}, \mathsf{NP}, \mathsf{VP}, \mathsf{V}\}$,
$\mathcal{R}_0 = \{\mathsf{S} \rightarrow \mathsf{NP}\ \mathsf{VP}, \mathsf{VP} \rightarrow \mathsf{V}\ \mathsf{NP}, \mathsf{NP} \rightarrow \mathsf{Sam}, \mathsf{NP} \rightarrow \mathsf{Sasha}, \mathsf{V} \rightarrow \mathsf{likes}, \mathsf{V} \rightarrow \mathsf{hates}\}$

S
$\Rightarrow$ NP VP
$\Rightarrow$ NP V NP
$\Rightarrow$ Sam V NP
$\Rightarrow$ Sam V Sasha
$\Rightarrow$ Sam likes Sasha

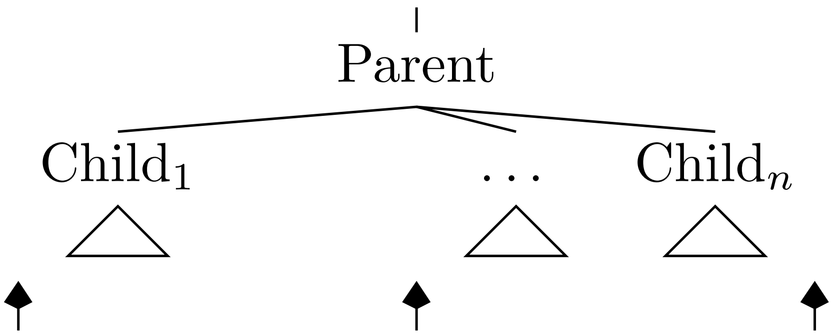
Steps in a *terminating derivation* are always *cuts in a parse tree*

Left-most and *right-most* derivations are *normal forms*



Enumerating trees and parsing strategies

A parsing strategy specifies the order in which nodes in trees are enumerated



Parsing strategy	Top-down	Left-corner	Bottom-up
Enumeration	Pre-order	In-order	Post-order
	Parent Child$_1$ . . . Child$_n$	Child$_1$ Parent . . . Child$_n$	Child$_1$ . . . Child$_n$ Parent

Top-down parses are left-most derivations

Leftmost derivation

S

S

Productions

S→ NP VP

NP→ D N

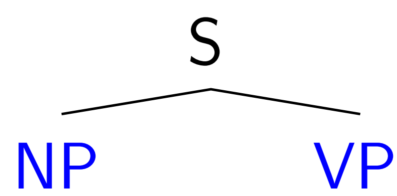
D→ no

N→ politican

VP→ V

V→ lies

Top-down parses are left-most derivations



Leftmost derivation

S

NP VP

Productions

S→ NP VP

NP→ D N

D→ no

N→ politican

VP→ V

V→ lies

Top-down parses are left-most derivations

Productions

S→ NP VP

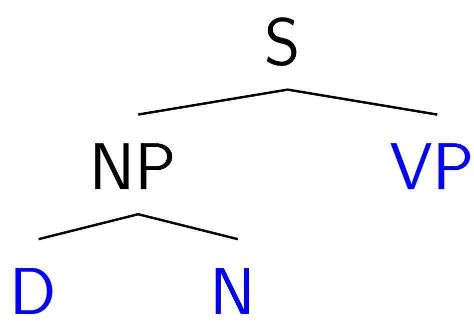
NP→ D N

D→ no

N→ politican

VP→ V

V→ lies



Leftmost derivation

S

NP VP

D N VP

Top-down parses are left-most derivations

Productions

S→ NP VP

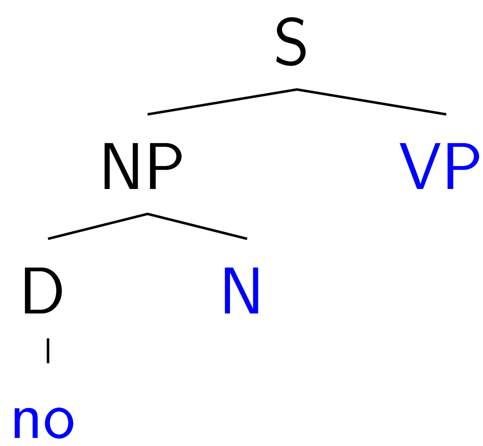
NP→ D N

D→ no

N→ politican

VP→ V

V→ lies



Leftmost derivation

S
NP VP
D N VP
no N VP

Top-down parses are left-most derivations

Productions

S→ NP VP

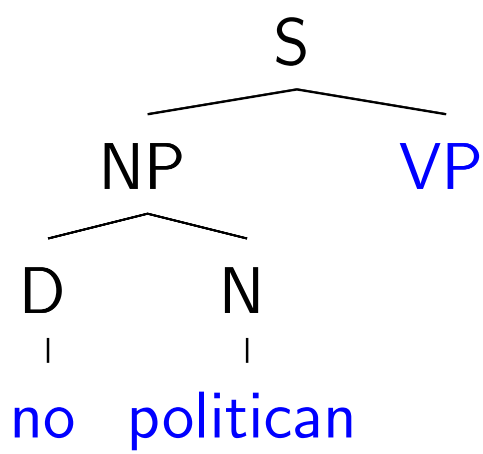
NP→ D N

D→ no

N→ politican

VP→ V

V→ lies



Leftmost derivation

S
NP VP
D N VP
no N VP
no politican VP

Top-down parses are left-most derivations

Productions

S→ NP VP

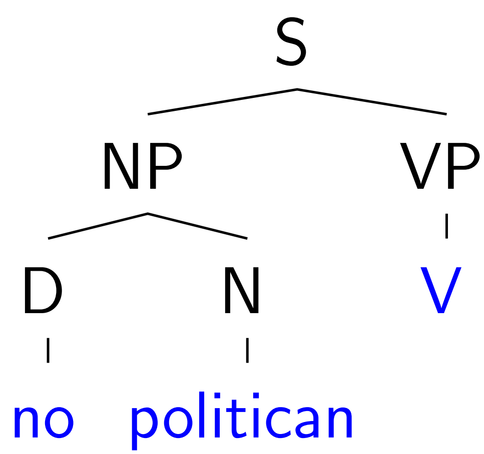
NP→ D N

D→ no

N→ politican

VP→ V

V→ lies



Leftmost derivation

S

NP VP

D N VP

no N VP

no politican VP

no politican V

Top-down parses are left-most derivations

Productions

S→ NP VP

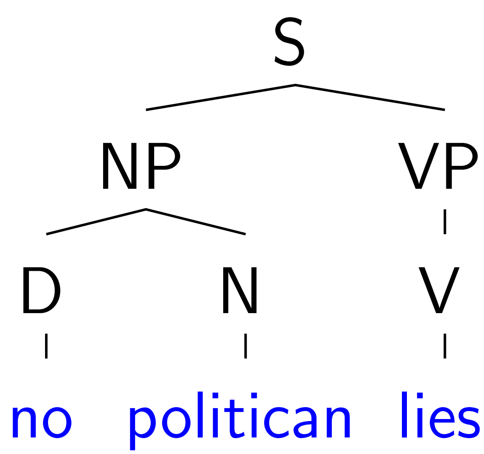
NP→ D N

D→ no

N→ politican

VP→ V

V→ lies



Leftmost derivation

S
NP VP
D N VP
no N VP
no politican VP
no politican V
no politican lies

Bottom-up parses are reversed right-most derivations

Rightmost derivation

Productions

S→ NP VP

NP→ D N

D→ no

N→ politican

VP→ V

V→ lies

no politican lies

no politican lies

Bottom-up parses are reversed right-most derivations

Rightmost derivation

Productions

S→ NP VP

NP→ D N

D→ no

N→ politican

VP→ V

V→ lies

$$\begin{array}{ccc} \text{D} & & \\ | & & \\ \text{no} & \text{politican} & \text{lies} \end{array}$$

D politican lies
no politican lies

Bottom-up parses are reversed right-most derivations

Rightmost derivation

Productions

S→ NP VP

NP→ D N

D→ no

N→ politican

VP→ V

V→ lies

D N

no politican lies

D N lies

D politican lies

no politican lies

Bottom-up parses are reversed right-most derivations

Rightmost derivation

Productions

S→ NP VP

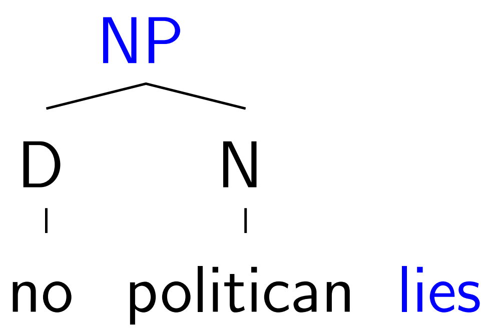
NP→ D N

D→ no

N→ politican

VP→ V

V→ lies



NP lies
D N lies
D politican lies
no politican lies

Bottom-up parses are reversed right-most derivations

Rightmost derivation

Productions

S→ NP VP

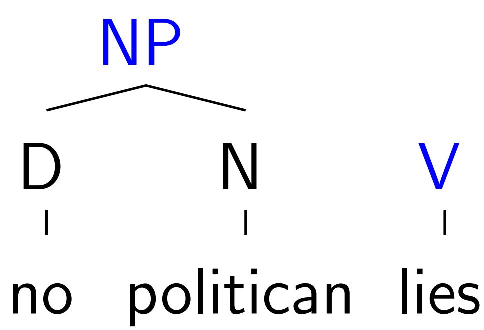
NP→ D N

D→ no

N→ politican

VP→ V

V→ lies



NP V
NP lies
D N lies
D politican lies
no politican lies

Bottom-up parses are reversed right-most derivations

Productions

S→ NP VP

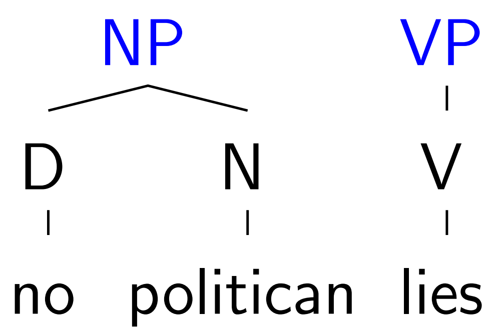
NP→ D N

D→ no

N→ politican

VP→ V

V→ lies



Rightmost derivation

NP VP
NP V
NP lies
D N lies
D politican lies
no politican lies

Bottom-up parses are reversed right-most derivations

Productions

S→ NP VP

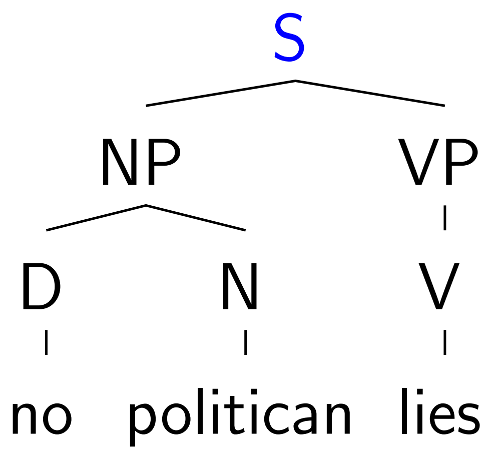
NP→ D N

D→ no

N→ politican

VP→ V

V→ lies



Rightmost derivation

S
NP VP
NP V
NP lies
D N lies
D politican lies
no politican lies

Probabilistic Context Free Grammars

A *Probabilistic Context Free Grammar* (PCFG) G consists of

- a CFG $(\mathcal{V}, \mathcal{S}, S, \mathcal{R})$ with no useless productions, and
- *production probabilities* $p(A \rightarrow \beta) = \mathrm{P}(\beta|A)$ for each $A \rightarrow \beta \in \mathcal{R}$, the conditional probability of an A expanding to β

A production $A \rightarrow \beta$ is *useless* iff it is not used in any terminating derivation, i.e., there are no derivations of the form $S \Rightarrow^\star \gamma A \delta \Rightarrow \gamma \beta \delta \Rightarrow^* w$ for any $\gamma, \delta \in (N \cup T)^\star$ and $w \in T^\star$.

If $r_1 \ldots r_n$ is a sequence of productions used to generate a tree ψ, then

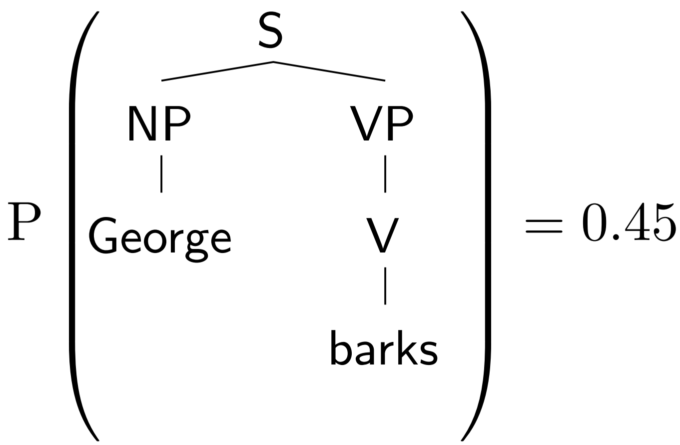
$$
\begin{aligned}
\mathrm{P}_G(\psi) &= p(r_1) \ldots p(r_n) \\
&= \prod_{r \in \mathcal{R}} p(r)^{f_r(\psi)}
\end{aligned}
$$

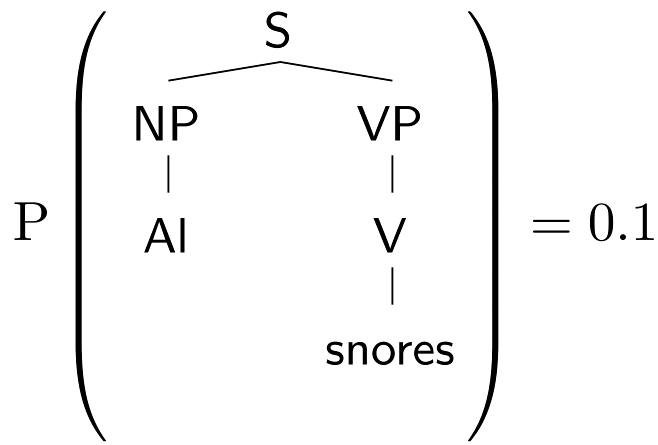
where $f_r(\psi)$ is the number of times r is used in deriving ψ

$\sum_\psi \mathrm{P}_G(\psi) = 1$ if p satisfies suitable constraints

Example PCFG

1.0	S $\longrightarrow$ NP VP	1.0	VP $\longrightarrow$ V
0.75	NP $\longrightarrow$ George	0.25	NP $\longrightarrow$ Al
0.6	V $\longrightarrow$ barks	0.4	V $\longrightarrow$ snores





Topics

- Graphical models and Bayes networks
- Markov chains and hidden Markov models
- (Probabilistic) context-free grammars
- *(Probabilistic) finite-state machines*
- Computation with PCFGs
- Estimation of PCFGs
- Lexicalized and bi-lexicalized PCFGs
- Non-local dependencies and log-linear models
- Stochastic unification-based grammars

Finite-state automata - Informal description

Finite-state automata are devices that generate arbitrarily long strings one symbol at a time.

At each step the automaton is in one of a finite number of states.

Processing proceeds as follows:

1. Initialize the machine's state s to the *start state* and $w = \epsilon$ (the empty string)
2. Loop:
   (a) Based on the current state s, decide whether to stop and return w
   (b) Based on the current state s, append a certain symbol x to w and update to s'

Mealy automata choose x based on s and s'

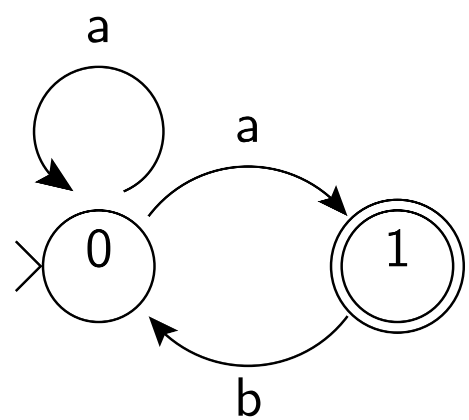
Moore automata (homogenous HMMs) choose x based on s' alone

Note: I'm simplifying here: Mealy and Moore *machines* are *transducers*
In probabilistic automata, these actions are directed by probability distributions

Mealy finite-state automata

Mealy automata emit terminals from arcs.

A *(Mealy) automaton* $M = (\mathcal{V}, \mathcal{S}, s_0, \mathcal{F}, \mathcal{M})$ consists of:



- $\mathcal{V}$, a set of *terminals*, ($\mathcal{V}_3 = \{\mathsf{a}, \mathsf{b}\}$)
- $\mathcal{S}$, a finite set of *states*, ($\mathcal{S}_3 = \{0, 1\}$)
- $s_0 \in \mathcal{S}$, the *start state*, ($s_{0_3} = 0$)
- $\mathcal{F} \subseteq \mathcal{S}$, the set of *final states* ($\mathcal{F}_3 = \{1\}$) and
- $\mathcal{M} \subseteq \mathcal{S} \times \mathcal{V} \times \mathcal{S}$, the *state transition relation*. ($\mathcal{M}_3 = \{(0, \mathsf{a}, 0), (0, \mathsf{a}, 1), (1, \mathsf{b}, 0)\}$)

A *accepting derivation* of a string $v_1 \ldots v_n \in \mathcal{V}^\star$ is a sequence of states $s_0 \ldots s_n \in \mathcal{S}^\star$ where:

- s_0 is the start state
- $s_n \in \mathcal{F}$, and
- for each $i = 1 \ldots n$, $(s_{i-1}, v_i, s_i) \in \mathcal{M}$.

00101 is an accepting derivation of aaba.

Probabilistic Mealy automata

A *probabilistic Mealy automaton* $M = (\mathcal{V}, \mathcal{S}, s_0, p_f, p_m)$ consists of:

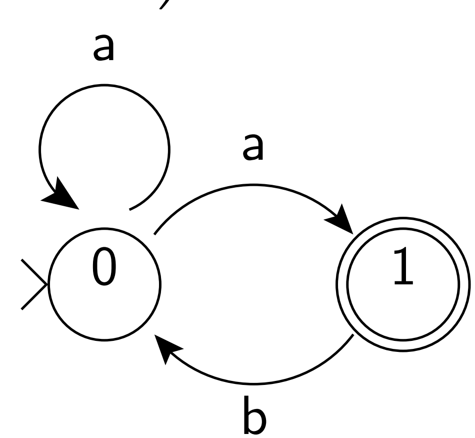
- terminals $\mathcal{V}$, states $\mathcal{S}$ and start state $s_0 \in \mathcal{S}$ as before,
- $p_f(s)$, the probability of *halting at state* $s \in \mathcal{S}$, and
- $p_m(v, s'|s)$, the probability of *moving from* $s \in \mathcal{S}$ *to* $s' \in \mathcal{S}$ *and emitting a* $v \in \mathcal{V}$.

where $p_f(s) + \sum_{v \in \mathcal{V}, s' \in \mathcal{S}} p_m(v, s'|s) = 1$ for all $s \in \mathcal{S}$ (halt or move on)

The probability of a derivation with states $s_0 \ldots s_n$ and outputs $v_1 \ldots v_n$ is:

$$\mathrm{P}_M(s_0 \ldots s_n; v_1 \ldots v_n) \;=\; \left(\prod_{i=1}^{n} p_m(v_i, s_i | s_{i-1}) \right) p_f(s_n)$$

Example: $p_f(\mathsf{0}) = 0, p_f(\mathsf{1}) = 0.1,$
$p_m(\mathsf{a}, \mathsf{0}|\mathsf{0}) = 0.2, p_m(\mathsf{a}, \mathsf{1}|\mathsf{0}) = 0.8, p_m(\mathsf{b}, \mathsf{0}|\mathsf{1}) = 0.9$
$P_M(\mathsf{00101}, \mathsf{aaba}) \;=\; 0.2 \times 0.8 \times 0.9 \times 0.8 \times 0.1$

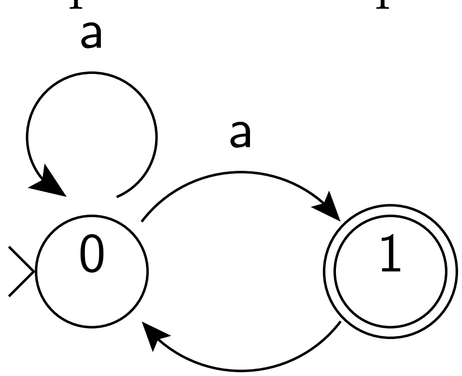


Bayes net representation of Mealy PFSA

In a Mealy automaton, the output is determined by the current and next state.

$$\begin{tikzcd}
\cdots \arrow[rr] & & S_{i-1} \arrow[rr] \arrow[dl] \arrow[dr] & & S_i \arrow[rr] \arrow[dl] \arrow[dr] & & S_{i+1} \arrow[rr] \arrow[dl] \arrow[dr] & & \cdots \\
& \cdots & & V_i & & V_{i+1} & & \cdots &
\end{tikzcd}$$

Example: state sequence 00101 for string aaba



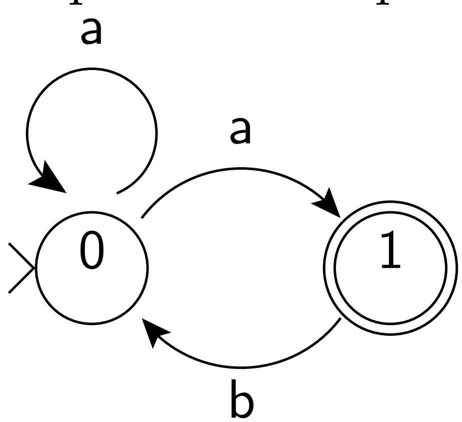
Mealy FSA

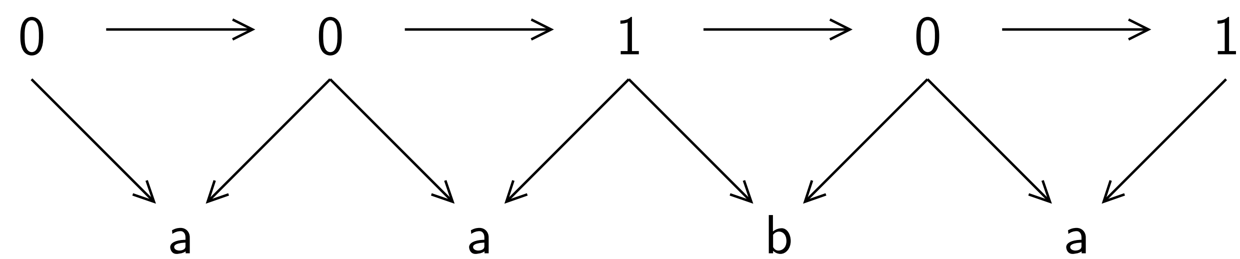
$$\begin{tikzcd}
0 \arrow[rr] \arrow[dr] & & 0 \arrow[rr] \arrow[dl] \arrow[dr] & & 1 \arrow[rr] \arrow[dl] \arrow[dr] & & 0 \arrow[rr] \arrow[dl] \arrow[dr] & & 1 \arrow[dl] \\
& \mathsf{a} & & \mathsf{a} & & \mathsf{b} & & \mathsf{a} &
\end{tikzcd}$$

Bayes net for aaba

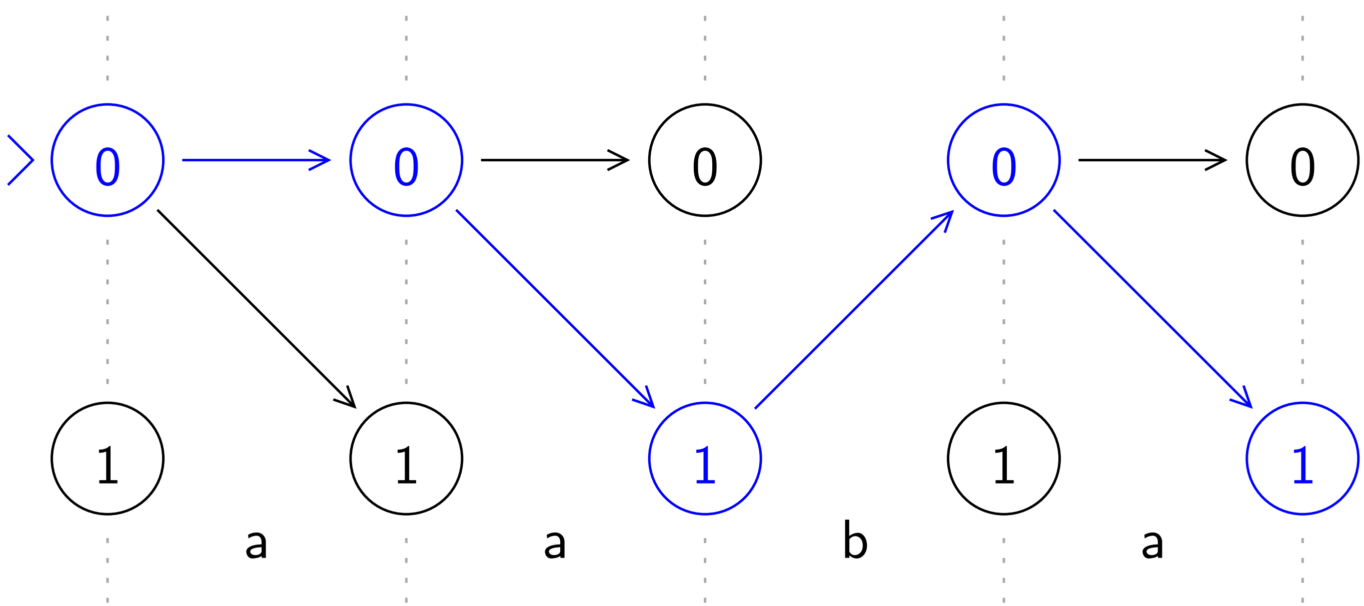
The trellis for a Mealy PFSA

Example: state sequence 00101 for string aaba





Bayes net for aaba

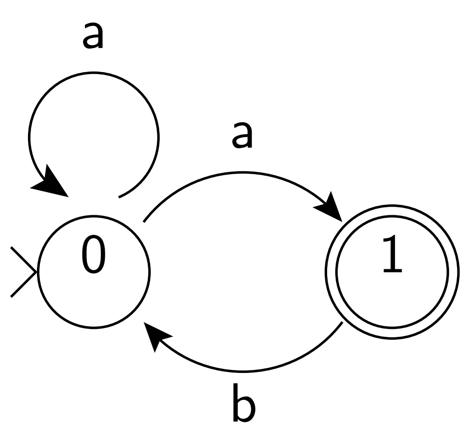


Probabilistic Mealy FSA as PCFGs

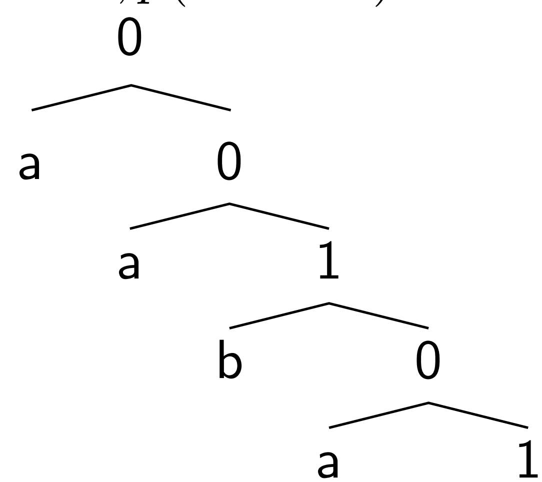
Given a Mealy PFSA $M = (\mathcal{V}, \mathcal{S}, s_0, p_f, p_m)$, let G_M have the same terminals, states and start state as M, and have productions

- $s \rightarrow \epsilon$ with probability $p_f(s)$ for all $s \in \mathcal{S}$
- $s \rightarrow v\, s'$ with probability $p_m(v, s'|s)$ for all $s, s' \in \mathcal{S}$ and $v \in \mathcal{V}$

$p(0 \rightarrow \mathsf{a}\ 0) = 0.2, p(0 \rightarrow \mathsf{a}\ 1) = 0.8, p(1 \rightarrow \epsilon) = 0.1, p(1 \rightarrow \mathsf{b}\ 0) = 0.9$



Mealy FSA



PCFG parse of aaba

The FSA graph depicts the machine (i.e., all strings it generates), while the CFG tree depicts the analysis of a single string.

Moore finite state automata

Moore machines emit terminals from states.

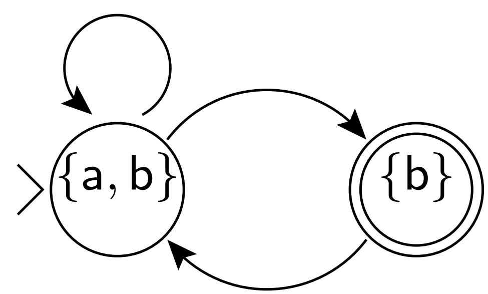
A *Moore finite state automaton* $M = (\mathcal{V}, \mathcal{S}, s_0, \mathcal{F}, \mathcal{M}, \mathcal{L})$ is composed of:

- $\mathcal{V}$, $\mathcal{S}$, s_0 and $\mathcal{F}$ are terminals, states, start state and final states as before
- $\mathcal{M} \subseteq \mathcal{S} \times \mathcal{S}$, the *state transition relation*
- $\mathcal{L} \subseteq \mathcal{S} \times \mathcal{V}$, the *state labelling function*

$(\mathcal{V}_4 = \{\mathsf{a}, \mathsf{b}\}, \mathcal{S}_4 = \{\mathsf{0}, \mathsf{1}\}, s_{0_4} = \mathsf{0}, \mathcal{F}_4 = \{\mathsf{1}\}, \mathcal{M}_4 = \{(\mathsf{0}, \mathsf{0}), (\mathsf{0}, \mathsf{1}), (\mathsf{1}, \mathsf{0})\},$
$\mathcal{L}_4 = \{(\mathsf{0}, \mathsf{a}), (\mathsf{0}, \mathsf{b}), (\mathsf{1}, \mathsf{b})\})$

A derivation of $v_1 \ldots v_n \in \mathcal{V}^\star$ is a sequence of states $s_0 \ldots s_n \in \mathcal{S}^\star$ where:

- s_0 is the start state, $s_n \in \mathcal{F}$,
- $(s_{i-1}, s_i) \in \mathcal{M}$, for $i = 1 \ldots n$
- $(s_i, v_i) \in \mathcal{L}$ for $i = 1 \ldots n$



0101 is an accepting derivation of bab

Probabilistic Moore automata

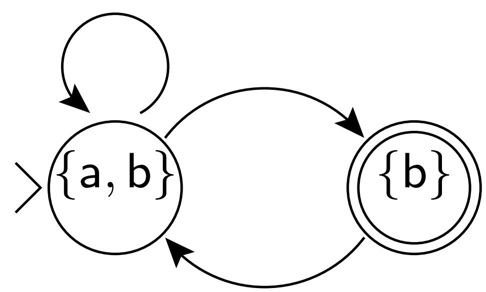
A *probabilistic Moore automaton* $M = (\mathcal{V}, \mathcal{S}, s_0, p_f, p_m, p_\ell)$ consists of:

- terminals $\mathcal{V}$, states $\mathcal{S}$ and start state $s_0 \in \mathcal{S}$ as before,
- $p_f(s)$, the probability of *halting at state* $s \in \mathcal{S}$,
- $p_m(s'|s)$, the probability of *moving from* $s \in \mathcal{S}$ *to* $s' \in \mathcal{S}$, and
- $p_\ell(v|s)$, the probability of *emitting* $v \in \mathcal{V}$ *from state* $s \in \mathcal{S}$.

where $p_f(s) + \sum_{s' \in \mathcal{S}} p_m(s'|s) = 1$ and $\sum_{v \in \mathcal{V}} p_\ell(v|s) = 1$ for all $s \in \mathcal{S}$.

The probability of a derivation with states $s_0 \ldots s_n$ and output $v_1 \ldots v_n$ is

$$\mathrm{P}_M(s_0 \ldots s_n; v_1 \ldots v_n) \;=\; \left(\prod_{i=1}^{n} p_m(s_i|s_{i-1}) p_\ell(v_i|s_i) \right) p_f(s_n)$$



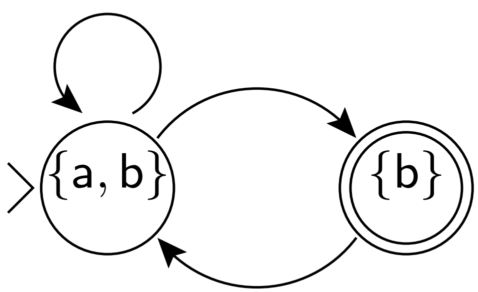
Example: $p_f(\mathsf{0}) = 0, p_f(\mathsf{1}) = 0.1$,
$p_\ell(\mathsf{a}|\mathsf{0}) = 0.4, p_\ell(\mathsf{b}|\mathsf{0}) = 0.6, p_\ell(\mathsf{b}|\mathsf{1}) = 1$,
$p_m(\mathsf{0}|\mathsf{0}) = 0.2, p_m(\mathsf{1}|\mathsf{0}) = 0.8, p_m(\mathsf{0}|\mathsf{1}) = 0.9$
$P_M(\mathsf{0101}, \mathsf{bab}) \;=\; (0.8 \times 1) \times (0.9 \times 0.4) \times (0.8 \times 1) \times 0.1$

Bayes net representation of Moore PFSA

In a Moore automaton, the output is determined by the current state, just as in an HMM (in fact, Moore automata are HMMs)

$$\begin{tikzcd}
\cdots \arrow[r] & S_{i-1} \arrow[r] \arrow[d] & S_i \arrow[r] \arrow[d] & S_{i+1} \arrow[r] \arrow[d] & \cdots \\
& V_{i-1} & V_i & V_{i+1} &
\end{tikzcd}$$

Example: state sequence 0101 for string bab



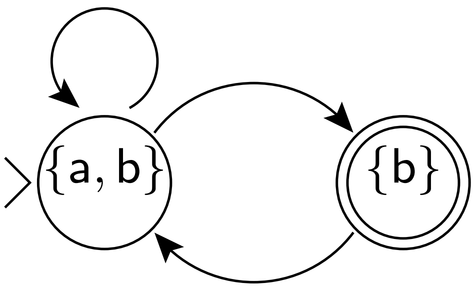
Moore FSA

$$\begin{tikzcd}
\mathsf{0} \arrow[r] & \mathsf{1} \arrow[r] \arrow[d] & \mathsf{0} \arrow[r] \arrow[d] & \mathsf{1} \arrow[d] \\
& \mathsf{b} & \mathsf{a} & \mathsf{b}
\end{tikzcd}$$

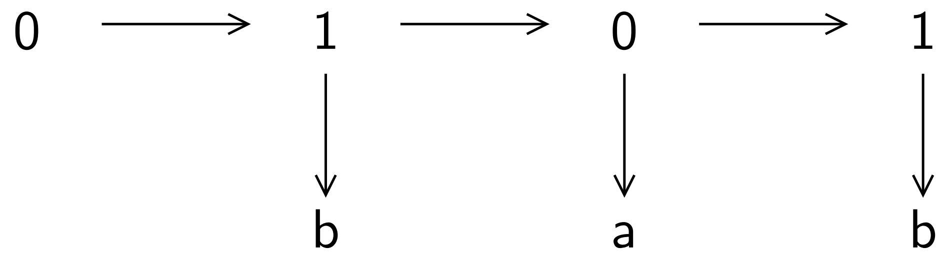
Bayes net for bab

Trellis representation of Moore PFSA

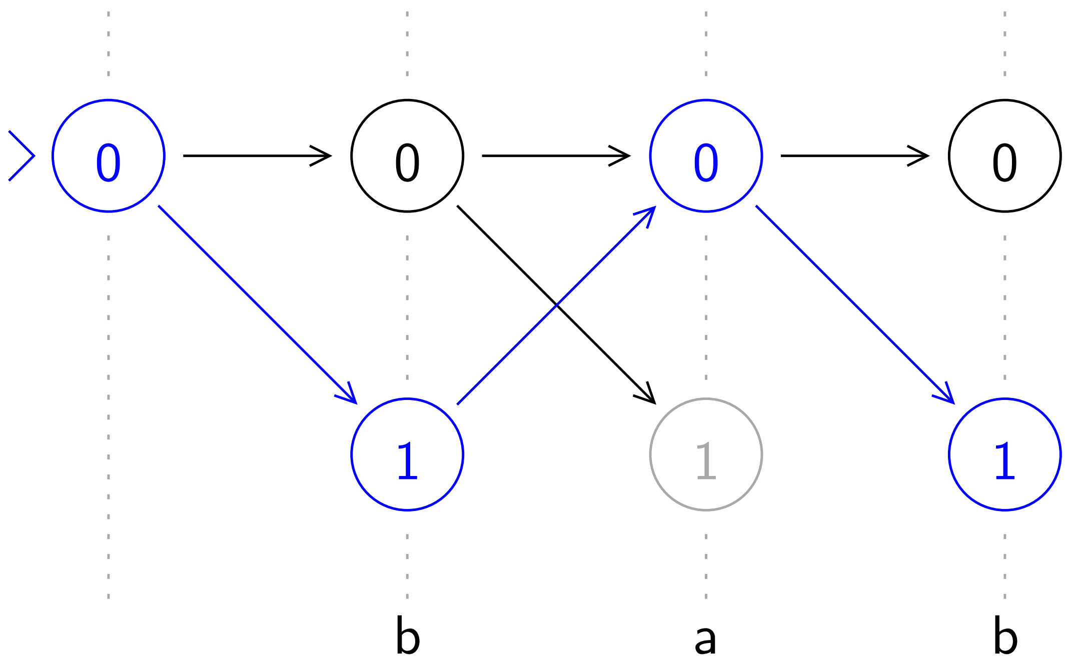
Example: state sequence 0101 for string bab



Moore FSA



Bayes net for bab

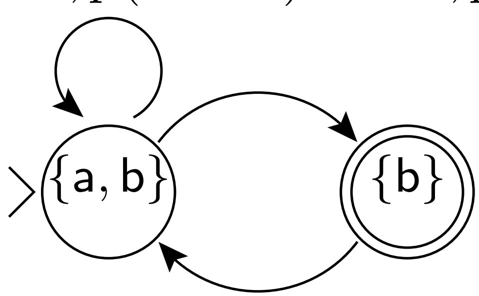


Probabilistic Moore FSA as PCFGs

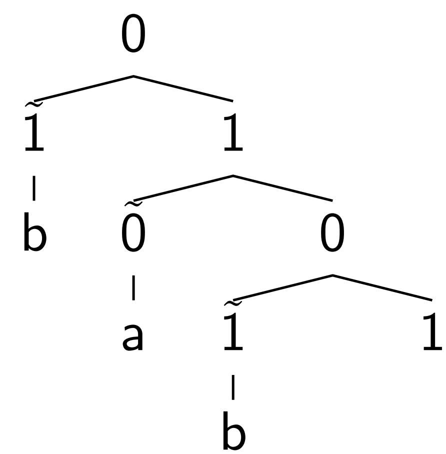
Given a Moore PFSA $M = (\mathcal{V}, \mathcal{S}, s_1, p_f, p_m, p_\ell)$, let G_M have the same terminals and start state as M, *two nonterminals s and $\tilde{s}$ for each state $s \in \mathcal{S}$*, and productions

- $s \rightarrow \tilde{s}'\, s'$ with probability $p_m(s'|s)$
- $s \rightarrow \epsilon$ with probability $p_f(s)$
- $\tilde{s} \rightarrow v$ with probability $p_\ell(v|s)$

$p(\mathsf{0} \rightarrow \tilde{\mathsf{0}}\,\mathsf{0}) = 0.2, p(\mathsf{0} \rightarrow \tilde{\mathsf{1}}\,\mathsf{1}) = 0.8,$
$p(\mathsf{1} \rightarrow \epsilon) = 0.1, p(\mathsf{1} \rightarrow \tilde{\mathsf{0}}\,\mathsf{0}) = 0.9,$
$p(\tilde{\mathsf{0}} \rightarrow \mathsf{a}) = 0.4, p(\tilde{\mathsf{0}} \rightarrow \mathsf{b}) = 0.6, p(\tilde{\mathsf{1}} \rightarrow \mathsf{b}) = 1$



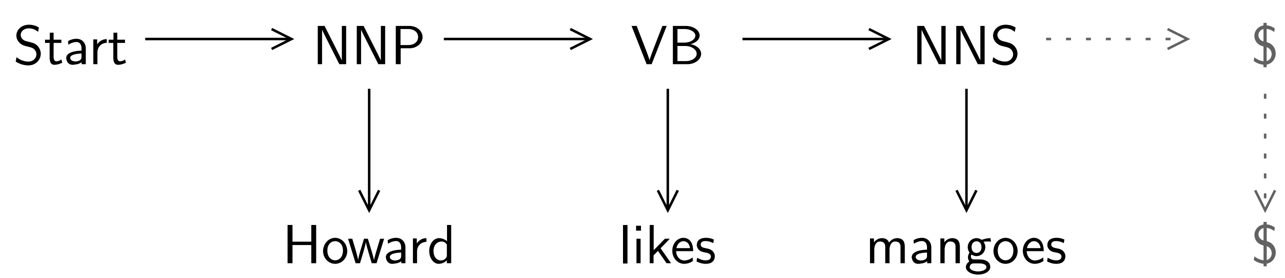
Moore FSA

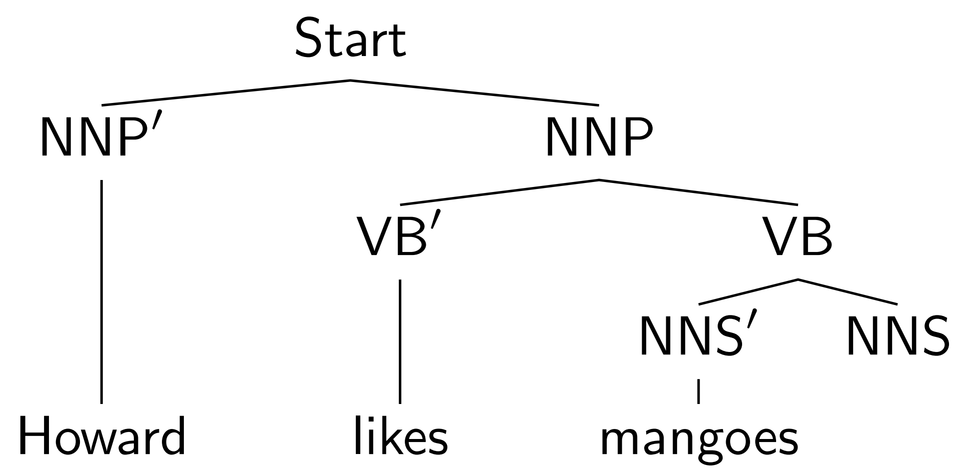


PCFG parse of bab

Bi-tag POS tagging

HMM or Moore PFSA whose states are POS tags





Mealy vs Moore automata

- Mealy automata emit terminals from arcs
  - a probabilistic Mealy automaton has $|\mathcal{V}||\mathcal{S}|^2 + |\mathcal{S}|$ parameters
- Moore automata emit terminals from states
  - a probabilistic Moore automaton has $(|\mathcal{V}| + 1)|\mathcal{S}|$ parameters

In a POS-tagging application, $|\mathcal{S}| \approx 50$ and $|\mathcal{V}| \approx 2 \times 10^4$

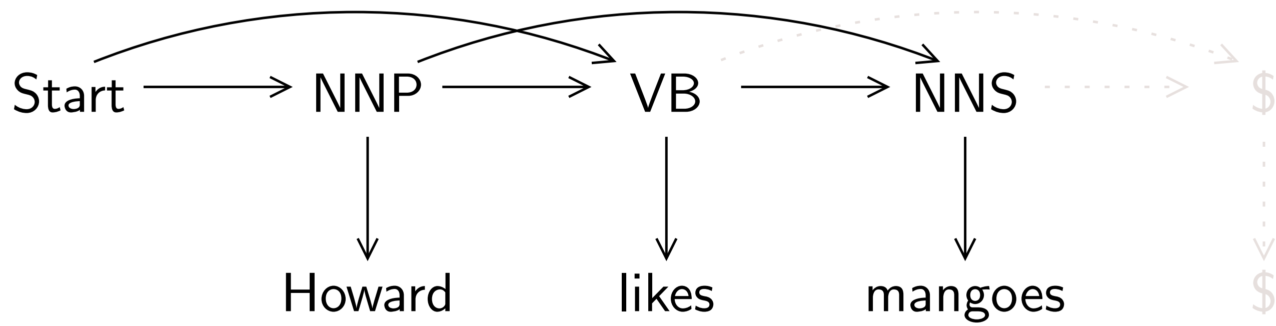
- A Mealy automaton has $\approx 5 \times 10^7$ parameters
- A Moore automaton has $\approx 10^6$ parameters

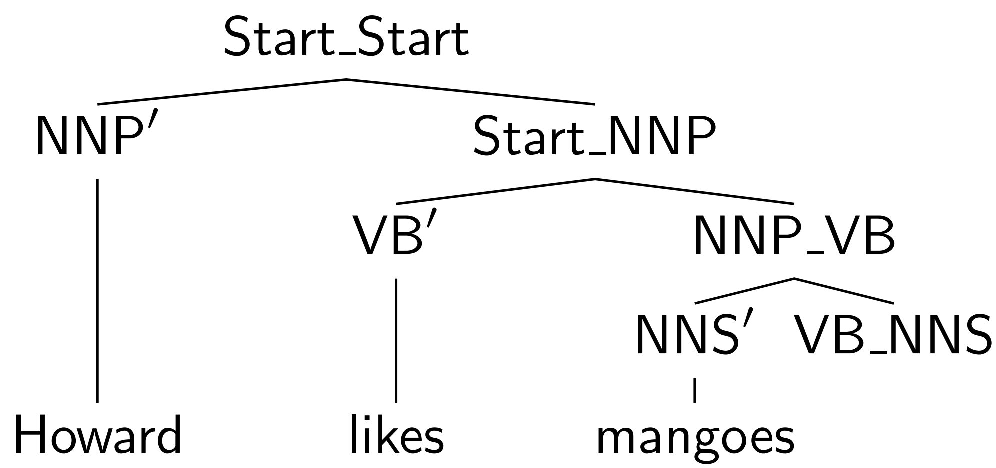
A Moore automaton seems more reasonable for POS-tagging

The number of parameters grows rapidly as the number of states grows

$\Rightarrow$ *Smoothing* is a practical necessity

Tri-tag POS tagging





Given a set of POS tags $\mathcal{T}$, the tri-tag PCFG has productions

$$t_0 t_1 \rightarrow t'_2 \;\; t_1 t_2 \qquad t' \rightarrow v$$

for all $t_0, t_1, t_2 \in \mathcal{T}$ and $v \in \mathcal{V}$

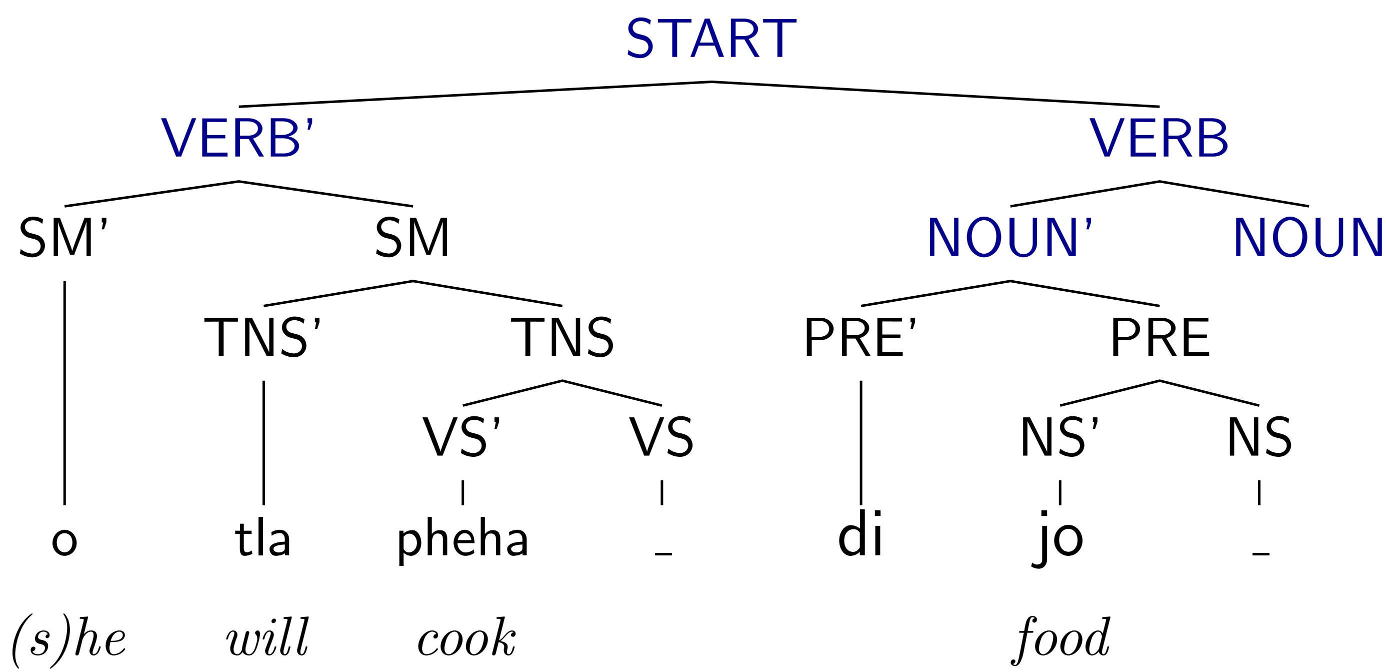
Advantages of using grammars

PCFGs provide a more flexible structural framework than HMMs and FSA

Sesotho is a Bantu language with rich agglutinative morphology

A *two-level HMM* seems appropriate:

- upper level generates a sequence of words, and
- lower level generates a sequence of morphemes in a word



Finite state languages and linear grammars

- The classes of all languages generated by Mealy and Moore FSA is the same. These languages are called *finite state languages*.
- The finite state languages are also generated by left-linear and by right-linear CFGs.
  - A CFG is *right linear* iff every production is of the form $A \rightarrow \beta$ or $A \rightarrow \beta\, B$ for $B \in \mathcal{S}$ and $\beta \in \mathcal{V}^\star$ (*nonterminals only appear at the end of productions*)
  - A CFG is *left linear* iff every production is of the form $A \rightarrow \beta$ or $A \rightarrow B\, \beta$ for $B \in \mathcal{S}$ and $\beta \in \mathcal{V}^\star$ (*nonterminals only appear at the beginning of productions*)
- The language ww^R, where $w \in \{\mathsf{a}, \mathsf{b}\}^\star$ and w^R is the reverse of w, is not a finite state language, but it is generated by a CFG
  $\Rightarrow$ some context-free languages are not finite state languages

Things you should know about FSA

- FSA are good ways of representing dictionaries and morphology
- Finite state *transducers* can encode phonological rules
- The finite state languages are closed under *intersection*, *union* and *complement*
- FSA can be *determinized* and *minimized*
- There are practical algorithms for computing these operations on large automata
- *All of this extends to probabilistic finite-state automata*
- *Much of this extends to PCFGs and tree automata*

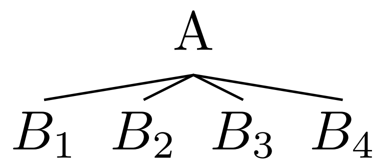
Topics

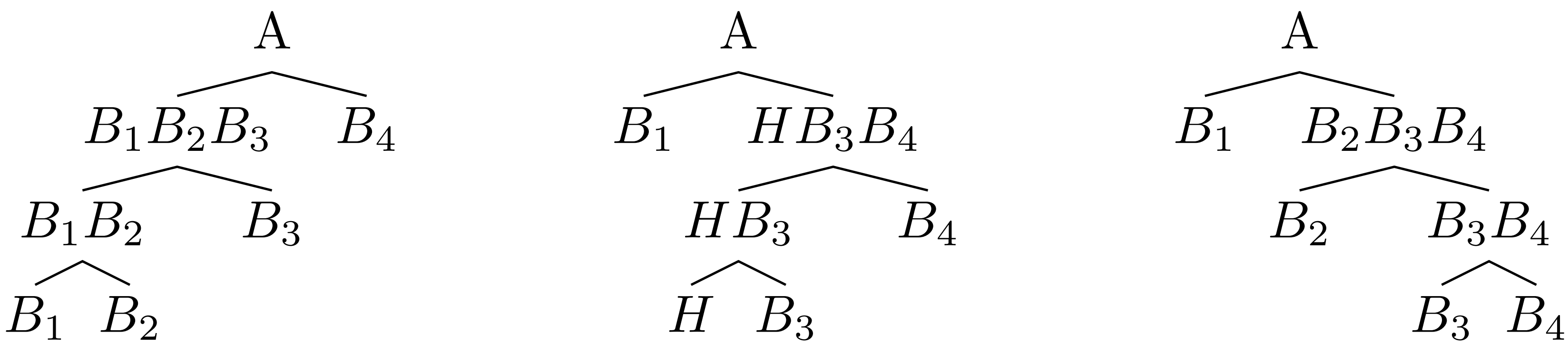
- Graphical models and Bayes networks
- Markov chains and hidden Markov models
- (Probabilistic) context-free grammars
- (Probabilistic) finite-state machines
- *Computation with PCFGs*
- Estimation of PCFGs
- Lexicalized and bi-lexicalized PCFGs
- Non-local dependencies and log-linear models
- Stochastic unification-based grammars

Binarization

Almost all efficient CFG parsing algorithms require productions have at most two children.

Binarization can be done as a preprocessing step, or implicitly during parsing.



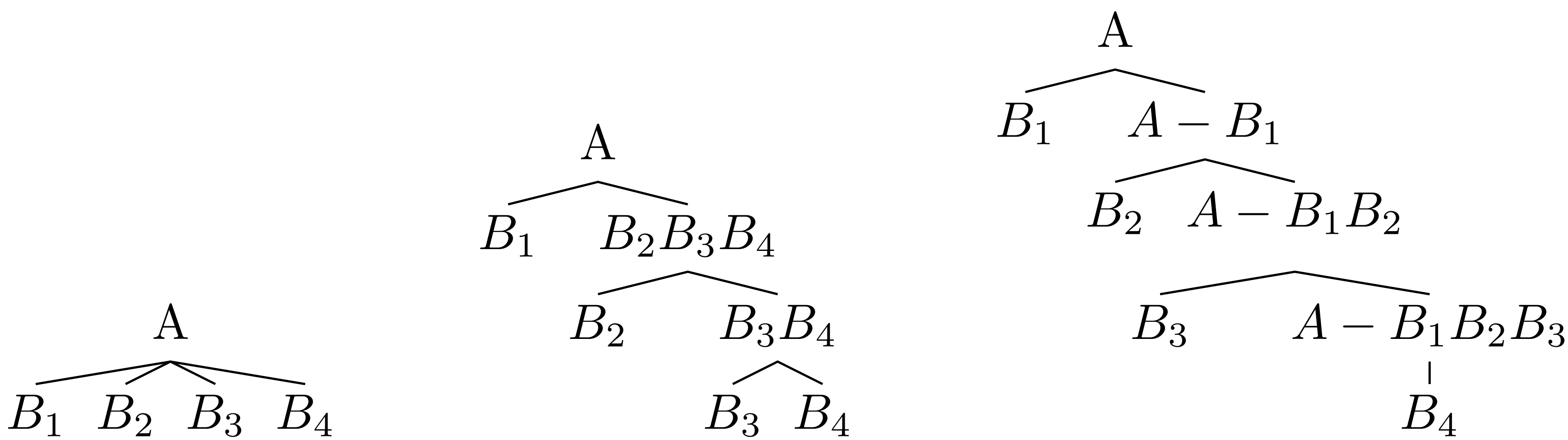


Left-factored　　Head-factored　　Right-factored

(assuming $H = B_2$)

More on binarization

- Binarization usually produces large numbers of new nonterminals
- These all appear in a certain position (e.g., end of production)
- *Design your parser loops and indexing so this is maximally efficient*
- Top-down and left-corner parsing benefit from specially designed binarization that *delays choice points as long as possible*



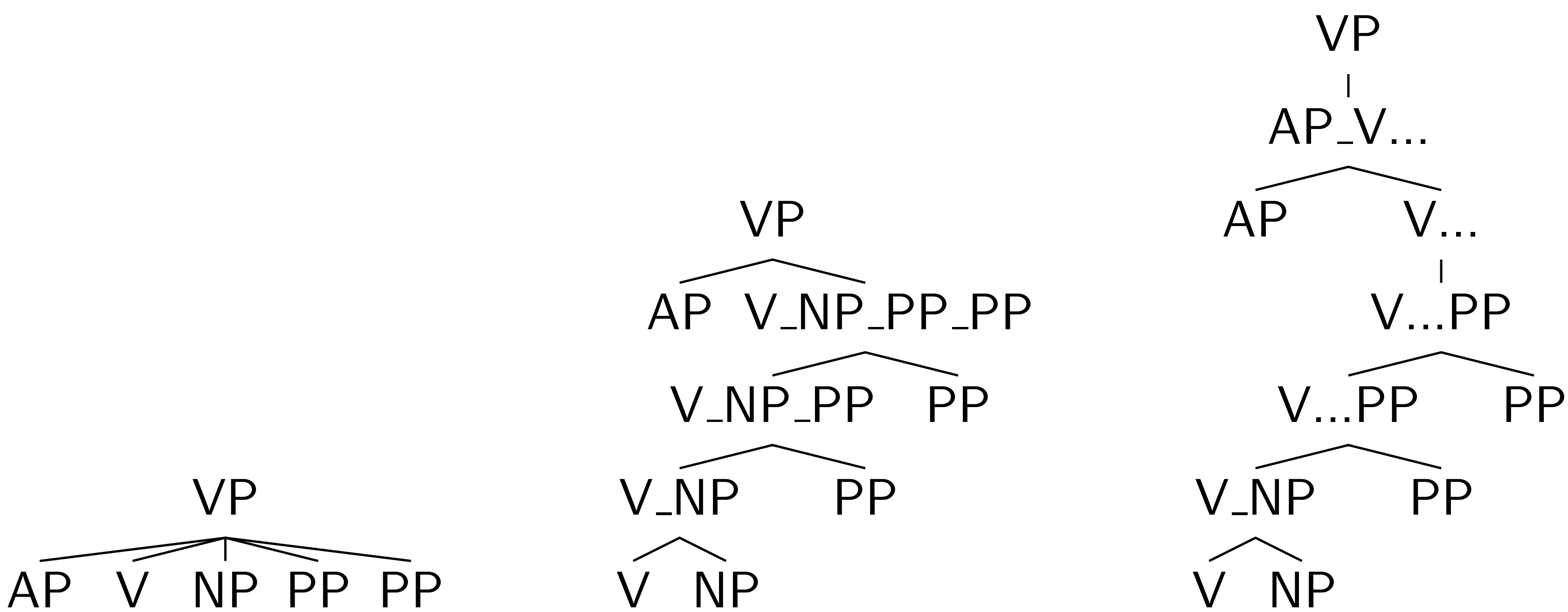
Unbinarized

Right-factored

Right-factored (top-down version)

Markov grammars

- Sometimes it can be desirable to smooth or generalize rules beyond what was actually observed in the treebank
- Markov grammars systematically "forget" part of the context



Unbinarized

Head-factored

(assuming $H = B_2$)

Markov grammar

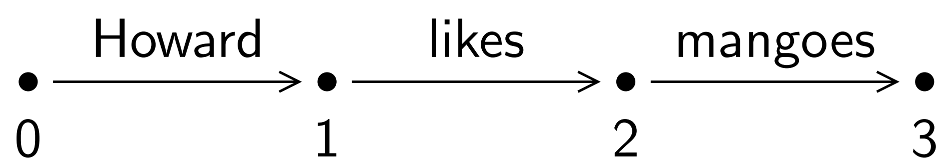
String positions

String positions are a systematic way of representing substrings in a string.

A *string position* of a string $w = x_1 \ldots x_n$ is an integer $0 \leq i \leq n$.

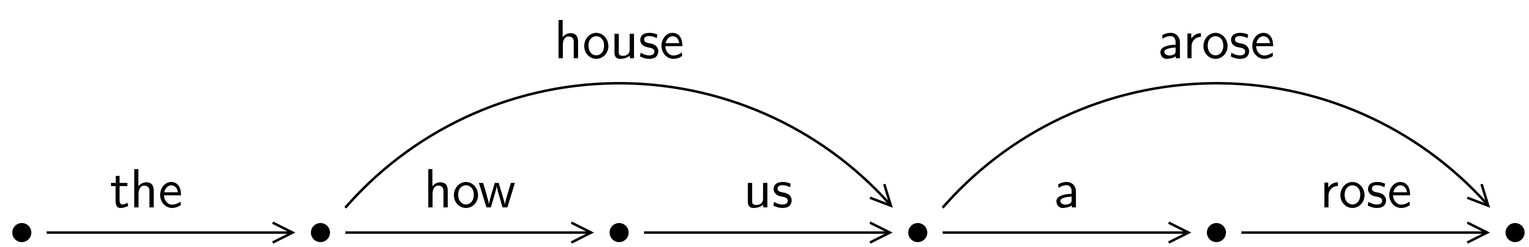
A substring of w is represented by a pair (i, j) of string positions, where $0 \leq i \leq j \leq n$.

$w_{i,j}$ represents the substring $w_{i+1} \ldots w_j$



Example: $w_{0,1} = \textsf{Howard}$, $w_{1,3} = \textsf{likes mangoes}$, $w_{1,1} = \epsilon$

- Nothing depends on string positions being numbers, so
- this all generalizes to speech recognizer *lattices*, which are graphs where vertices correspond to word boundaries



Dynamic programming computation

Assume $G = (\mathcal{V}, \mathcal{S}, s, \mathcal{R}, p)$ is in *Chomsky Normal Form*, i.e., all productions are of the form $A \rightarrow B\ C$ or $A \rightarrow x$, where $A, B, C \in \mathcal{S}$, $x \in \mathcal{V}$.

Goal: To compute $\mathrm{P}(w) = \sum_{\psi \in \Psi_G(w)} \mathrm{P}(\psi) = \mathrm{P}(s \Rightarrow^\star w)$

Data structure: A table $\mathrm{P}(A \Rightarrow^\star w_{i,j})$ for $A \in \mathcal{S}$ and $0 \leq i < j \leq n$

Base case: $\mathrm{P}(A \Rightarrow^\star w_{i-1,i}) \ = \ p(A \rightarrow w_{i-1,i})$ for $i = 1, \ldots, n$

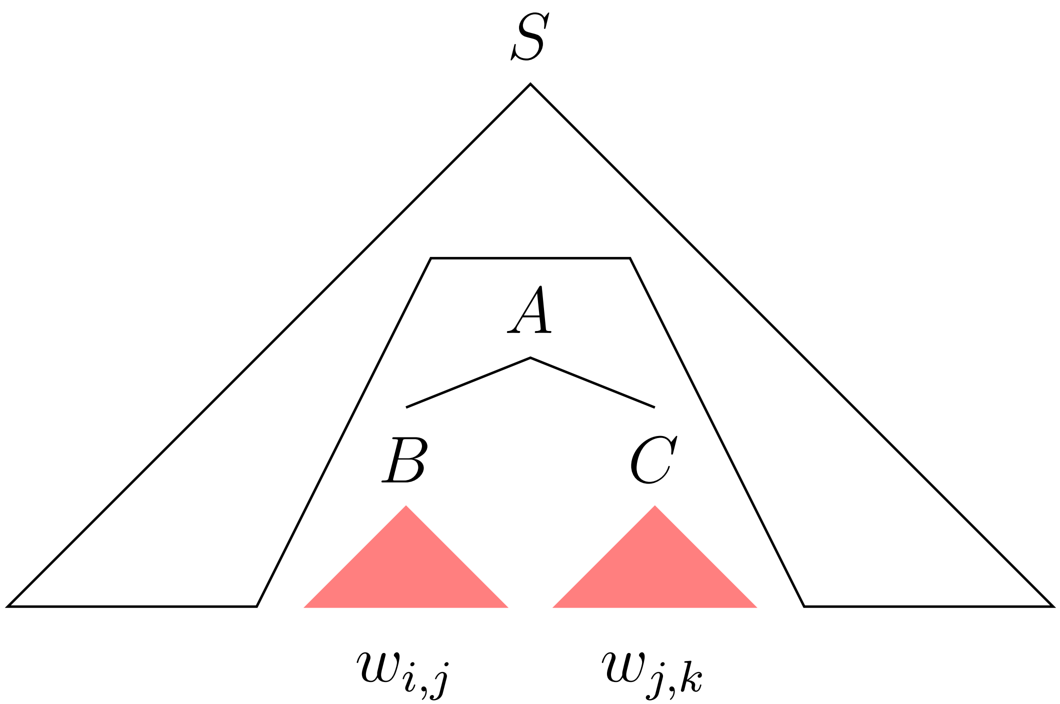
Recursion: $\mathrm{P}(A \Rightarrow^\star w_{i,k})$

$$= \sum_{j=i+1}^{k-1} \ \sum_{A \rightarrow B\, C \in \mathcal{R}(A)} p(A \rightarrow B\, C) \mathrm{P}(B \Rightarrow^* w_{i,j}) \mathrm{P}(C \Rightarrow^* w_{j,k})$$

Return: $\mathrm{P}(s \Rightarrow^\star w_{0,n})$

Dynamic programming recursion

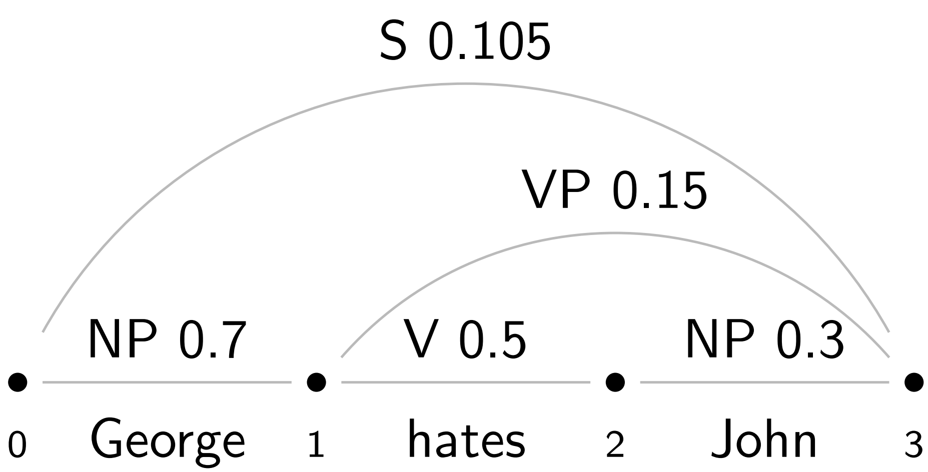
$$\mathrm{P}_G(A \Rightarrow^* w_{i,k}) = \sum_{j=i+1}^{k-1} \sum_{A \rightarrow B\,C \in \mathcal{R}(A)} p(A \rightarrow B\,C)\mathrm{P}_G(B \Rightarrow^* w_{i,j})\mathrm{P}_G(C \Rightarrow^* w_{j,k})$$



$\mathrm{P}_G(A \Rightarrow^* w_{i,k})$ is called an "*inside probability*".

Example PCFG parse

1.0	S → NP VP	1.0	VP → V NP
0.7	NP → *George*	0.3	NP → *John*
0.5	V → *likes*	0.5	V → *hates*



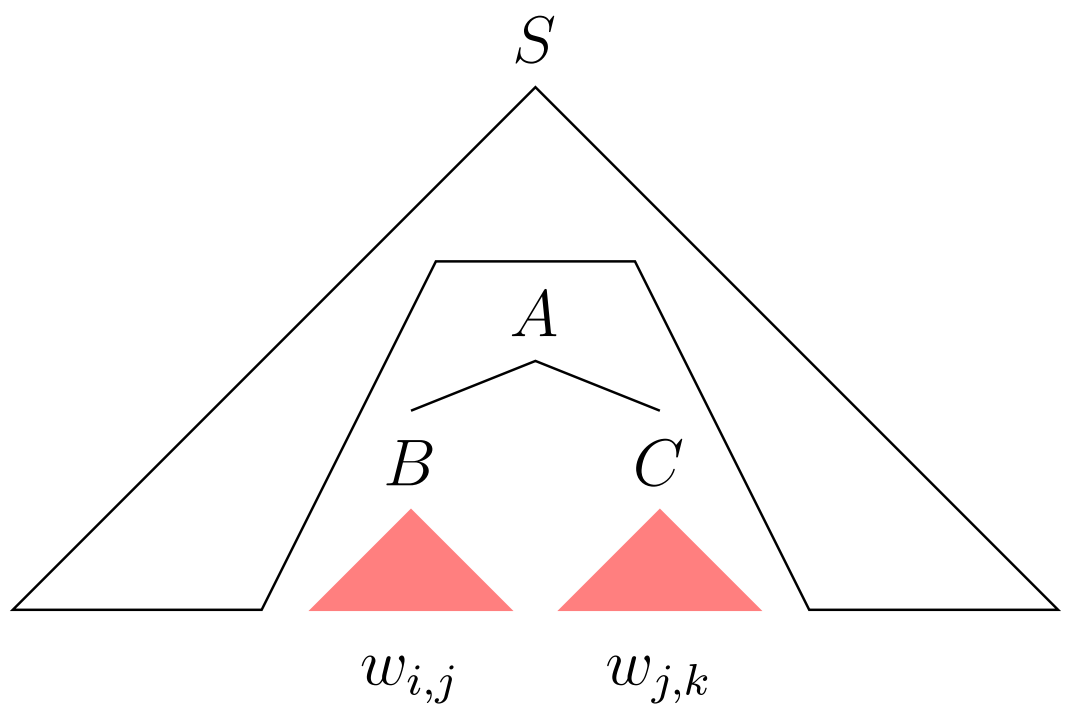
Right string position

Left string position	1	2	3
0	NP 0.7		S 0.105
1		V 0.5	VP 0.15
2			NP 0.3

CFG Parsing takes $n^3|\mathcal{R}|$ time

$$
\begin{aligned}
&\mathrm{P}_G(A \Rightarrow^* w_{i,k}) \\
&\quad = \sum_{j=i+1}^{k-1} \sum_{A \to B\, C \in \mathcal{R}(A)} p(A \to B\, C)\mathrm{P}_G(B \Rightarrow^* w_{i,j})\mathrm{P}_G(C \Rightarrow^* w_{j,k})
\end{aligned}
$$

The algorithm iterates over all rules $\mathcal{R}$ and all triples of string positions $0 \leq i < j < k \leq n$ (there are $n(n-1)(n-2)/6 = O(n^3)$ such triples)



PFSA parsing takes $n|\mathcal{R}|$ time

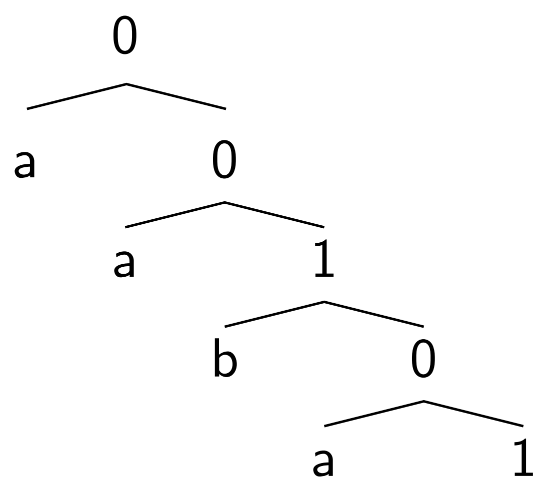
Because FSA trees are *uniformly right branching*,

- All non-trivial constituents end at the right edge of the sentence

$\Rightarrow$ The inside algorithm takes $n|\mathcal{R}|$ time

$$
\begin{aligned}
& \mathrm{P}_G(A \Rightarrow^* w_{i,n}) \\
& \quad = \sum_{A \to B\, C \in \mathcal{R}(A)} p(A \to B\, C) \mathrm{P}_G(B \Rightarrow^* w_{i,i+1}) \mathrm{P}_G(C \Rightarrow^* w_{i+1,n})
\end{aligned}
$$

- The standard FSM algorithms are just CFG algorithms, restricted to right-branching structures



Unary productions and unary closure

Dealing with "one level" unary productions $A \to B$ is easy, but how do we deal with "loopy" unary productions $A \Rightarrow^{+} B \Rightarrow^{+} A$?

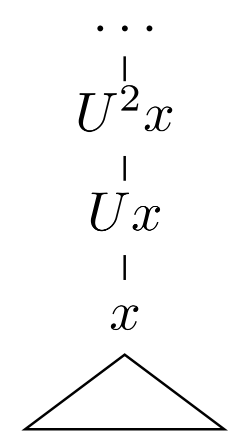
The *unary closure matrix* is $C_{ij} = \mathrm{P}(A_i \Rightarrow^{\star} A_j)$ for all $A_i, A_j \in \mathcal{S}$

Define $U_{ij} = p(A_i \to A_j)$ for all $A_i, A_j \in \mathcal{S}$

If x is a (column) vector of inside weights, Ux is a vector of the inside weights of parses with one unary branch above x

The *unary closure* is the sum of the inside weights with any number of unary branches:

$$\begin{aligned} x + Ux + U^2x + \ldots &= (1 + U + U^2 + \ldots)\, x \\ &= (1 - U)^{-1} x \end{aligned}$$



The unary closure matrix $C = (1 - U)^{-1}$ can be pre-computed, so unary closure is just a matrix multiplication.

Because "new" nonterminals introduced by binarization never occur in unary chains, unary closure is (relatively) cheap.

Finding the most likely parse of a string

Given a string $w \in \mathcal{V}^\star$, find the most likely tree $\widehat{\psi} = \arg\max_{\psi \in \Psi_G(w)} \mathrm{P}_G(\psi)$

(The most likely parse is also known as the *Viterbi parse*).

Claim: *If we substitute "max" for "+" in the algorithm for* $\mathrm{P}_G(w)$*, it returns* $\mathrm{P}_G(\widehat{\psi})$.

$$\mathrm{P}_G(\widehat{\psi}_{A,i,k}) = \max_{j=i+1,\ldots,k-1} \max_{A \to B\,C \in \mathcal{R}(A)} p(A \to B\,C) \mathrm{P}_G(\widehat{\psi}_{B,i,j}) \mathrm{P}_G(\widehat{\psi}_{C,j,k})$$

To return $\widehat{\psi}$, add "back-pointers" to keep track of best parse $\widehat{\psi}_{A,i,j}$ for each $A \Rightarrow^\star w_{i,j}$

Implementation note: There's no need to actually build these trees $\widehat{\psi}_{A,i,k}$; rather, the back-pointers in each table entry point to the table entries for the best parse's children

Semi-ring of rule weights

Our algorithms don't actually require that the values associated with productions are probabilities ...

Our algorithms only require that productions have values in some *semi-ring* with operations "$\oplus$" and "$\otimes$" with the usual associative and distributive laws

$\oplus$	$\otimes$	
$+$	$\times$	sum of probabilities or weights
max	$\times$	Viterbi parse
max	$+$	Viterbi parse with log probabilities
$\wedge$	$\vee$	Categorical CFG parsing

Topics

- Graphical models and Bayes networks
- Markov chains and hidden Markov models
- (Probabilistic) context-free grammars
- (Probabilistic) finite-state machines
- Computation with PCFGs
- *Estimation of PCFGs*
- Lexicalized and bi-lexicalized PCFGs
- Non-local dependencies and log-linear models
- Stochastic unification-based grammars

Two approaches to computational linguistics

"Rationalist": Linguist formulates generalizations and expresses them in a grammar

"Empiricist": Collect a corpus of examples, linguists annotate them with relevant information, a machine learning algorithm extracts generalizations

- I don't think there's a deep philosophical difference here, but many people do
- Continuous models do much better than categorical models (statistical inference uses more information than categorical inference)
- Humans are lousy at estimating numerical probabilities, but luckily parameter estimation is the one kind of machine learning that (sort of) works

Treebanks, prop-banks and discourse banks

- A *treebank* is a *corpus* of phrase structure trees
  - The *Penn treebank* consists of about a million words from the Wall Street Journal, or about 40,000 trees.
  - The *Switchboard corpus* consists of about a million words of treebanked spontaneous conversations, linked up with the acoustic signal.
  - Treebanks are being constructed for other languages also
- The Penn treebank is being annotated with *predicate argument structure* (PropBank) and *discourse relations*.

Maximum likelihood estimation

An *estimator* $\hat{p}$ for parameters $p \in \mathcal{P}$ of a model $\mathrm{P}_p(X)$ is a function from data D to $\hat{p}(D) \in \mathcal{P}$.

The *likelihood* $L_D(p)$ and *log likelihood* $\ell_D(p)$ of data $D = (x_1 \ldots x_n)$ with respect to model parameters p is:

$$\begin{aligned} L_D(p) &= \mathrm{P}_p(x_1) \ldots \mathrm{P}_p(x_n) \\ \ell_D(p) &= \sum_{i=1}^{n} \log \mathrm{P}_p(x_i) \end{aligned}$$

The *maximum likelihood estimate* (MLE) $\hat{p}_{\mathrm{MLE}}$ of p from D is:

$$\hat{p}_{\mathrm{MLE}} = \arg\max_p L_D(p) = \arg\max_p \ell_D(p)$$

Optimization and Lagrange multipliers

$\partial f(x)/\partial x = 0$ at the *unconstrained optimum* of $f(x)$

But maximum likelihood estimation often requires optimizing $f(x)$ subject to constraints $g_k(x) = 0$ for $k = 1, \ldots, m$.

Introduce *Lagrange multipliers* $\lambda = (\lambda_1, \ldots, \lambda_m)$, and define:

$$F(x, \lambda) \;=\; f(x) - \lambda \cdot g(x) \;=\; f(x) - \sum_{k=1}^{m} \lambda_k g_k(x)$$

Then at the constrained optimum, all of the following hold:

$$\begin{aligned} 0 \;&=\; \partial F(x, \lambda)/\partial x \;=\; \partial f(x)/\partial x - \sum_{k=1}^{m} \lambda_k \partial g_k(x)/\partial x \\ 0 \;&=\; \partial F(x, \lambda)/\partial \lambda \;=\; g(x) \end{aligned}$$

Biased coin example

Model has parameters $p = (p_h, p_t)$ that satisfy constraint $p_h + p_t = 1$.

Log likelihood of data $D = (x_1, \ldots, x_n)$, $x_i \in \{h, t\}$, is

$$\ell_D(p) \;=\; \log(p_{x_1} \ldots p_{x_n}) \;=\; n_h \log p_h + n_t \log p_t$$

where n_h is the number of h in D, and n_t is the number of t in D.

$$\begin{aligned} F(p, \lambda) &= n_h \log p_h + n_t \log p_t - \lambda(p_h + p_t - 1) \\ 0 &= \partial F / \partial p_h \;=\; n_h / p_h - \lambda \\ 0 &= \partial F / \partial p_t \;=\; n_t / p_t - \lambda \end{aligned}$$

From the constraint $p_h + p_t = 1$ and the last two equations:

$$\begin{aligned} \lambda &= n_h + n_t \\ p_h &= n_h / \lambda \;=\; n_h / (n_h + n_t) \\ p_t &= n_t / \lambda \;=\; n_t / (n_h + n_t) \end{aligned}$$

So *the MLE is the relative frequency*

PCFG MLE from visible data

Data: A *treebank* of parse trees $D = \psi_1, \ldots, \psi_n$.

$$\ell_D(p) \;=\; \sum_{i=1}^{n} \log \mathrm{P}_G(\psi_i) \;=\; \sum_{A\to\alpha\in\mathcal{R}} n_{A\to\alpha}(D) \log p(A\to\alpha)$$

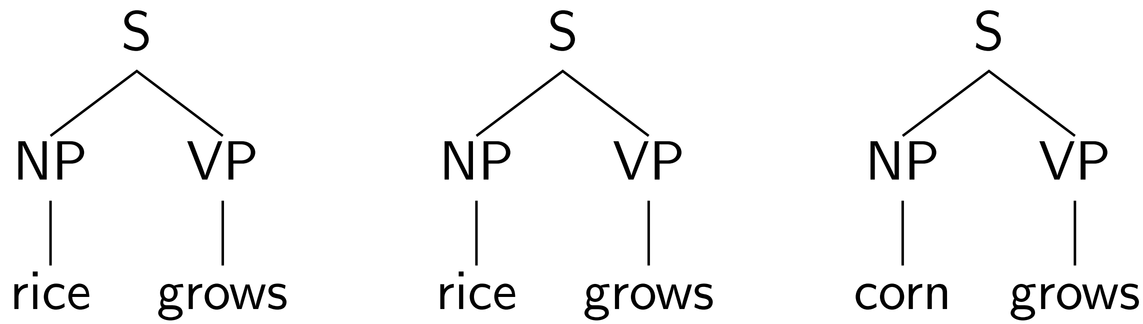
Introduce $|\mathcal{S}|$ Lagrange multipliers $\lambda_B, B \in \mathcal{S}$ for the constraints $\sum_{B\to\beta\in\mathcal{R}(B)} p(B\to\beta) = 1$. Then:

$$\frac{\partial\left(\ell(p) - \sum_{B\in\mathcal{S}} \lambda_B \left(\sum_{B\to\beta\in\mathcal{R}(B)} p(B\to\beta) - 1\right)\right)}{\partial p(A\to\alpha)} \;=\; \frac{n_{A\to\alpha}(D)}{p(A\to\alpha)} - \lambda_A$$

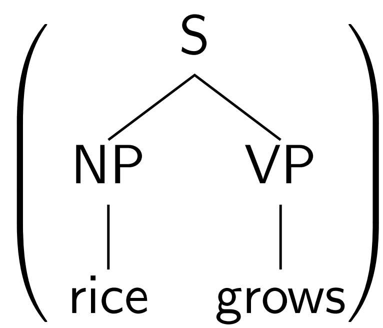
Setting this to 0, $\quad p(A\to\alpha) \;=\; \dfrac{n_{A\to\alpha}(D)}{\sum_{A\to\alpha'\in\mathcal{R}(A)} n_{A\to\alpha'}(D)}$

So the MLE for PCFGs is the *relative frequency estimator*

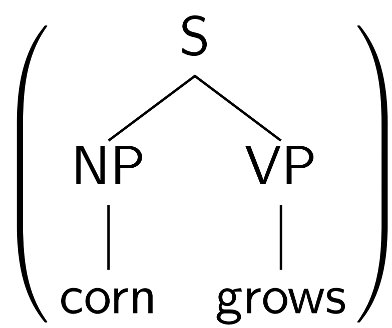
Example: Estimating PCFGs from visible data



Rule	Count	Rel Freq
S $\longrightarrow$ NP VP	3	1
NP $\longrightarrow$ rice	2	2/3
NP $\longrightarrow$ corn	1	1/3
VP $\longrightarrow$ grows	3	1

P 

$= 2/3$

P 

$= 1/3$

Properties of MLE

- *Consistency:* As the sample size grows, the estimates of the parameters converge on the true parameters
- *Asymptotic optimality:* For large samples, there is no other consistent estimator whose estimates have lower variance
- The MLEs for statistical grammars work well in practice.
  - The Penn Treebank has $\approx$ 1.2 million words of Wall Street Journal text annotated with syntactic trees
  - The PCFG estimated from the Penn Treebank has $\approx$ 15,000 rules

PCFG estimation from hidden data

Data: A corpus of sentences $D' = w_1, \ldots, w_n$.

$$\ell_{D'}(p) = \sum_{i=1}^{n} \log \mathrm{P}_G(w_i). \qquad \mathrm{P}_G(w) = \sum_{\psi \in \Psi_G(w)} \mathrm{P}_G(\psi).$$

$$\frac{\partial \ell_{D'}(p)}{\partial p(A \to \alpha)} = \frac{\sum_{i=1}^{n} \mathrm{E}_G[n_{A\to\alpha} | w_i]}{p(A \to \alpha)}$$

where the *expected number of times $A \to \alpha$ is used in the parses of w is*:

$$\mathrm{E}_G[n_{A\to\alpha} | w] = \sum_{\psi \in \Psi_G(w)} n_{A\to\alpha}(\psi) \mathrm{P}_G(\psi | w).$$

Setting $\partial \ell_{D'} / \partial p(A \to \alpha)$ to the Lagrange multiplier λ_A and imposing the constraint $\sum_{B \to \beta \in \mathcal{R}(B)} p(B \to \beta) = 1$ yields:

$$p(A \to \alpha) = \frac{\sum_{i=1}^{n} \mathrm{E}_G[n_{A\to\alpha} | w_i]}{\sum_{A \to \alpha' \in \mathcal{R}(A)} \sum_{i=1}^{n} \mathrm{E}_G[n_{A\to\alpha'} | w_i]}$$

This is an iteration of the *expectation maximization* algorithm!

Expectation maximization

EM is a general technique for approximating the MLE when estimating parameters p from the *visible data* x is difficult, but estimating p from augmented data $z = (x, y)$ is easier (y is the *hidden data*).

The EM algorithm given visible data x:

1. guess initial value p_0 of parameters
2. repeat for $i = 0, 1, \ldots$ until convergence:

   Expectation step: For all $y_1, \ldots, y_n \in \mathcal{Y}$, generate pseudo-data $(x, y_1), \ldots, (x, y_n)$, where (x, y_j) has frequency $\mathrm{P}_{p_i}(y_j|x)$

   Maximization step: Set p_{i+1} to the MLE from the pseudo-data

The likelihood $\mathrm{P}_p(x)$ of the visible data x stays the same or increases on each iteration.

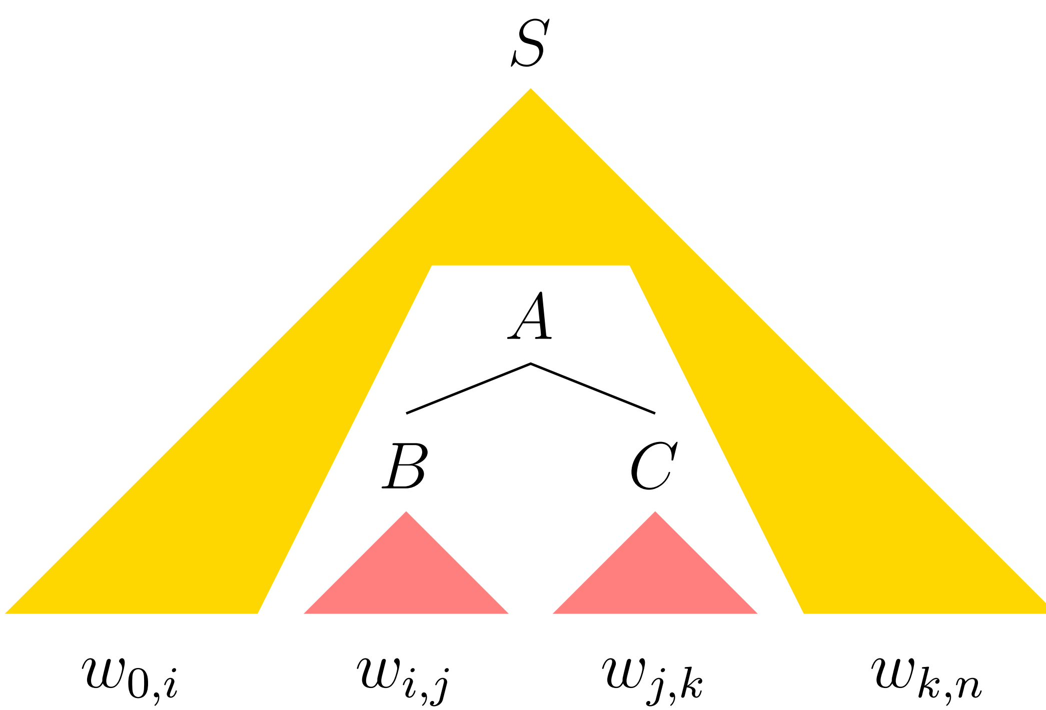
Sometimes it is not necessary to explicitly generate the pseudo-data (x, y); often it is possible to perform the maximization step directly from *sufficient statistics* (for PCFGs, the expected production frequencies)

Dynamic programming for expected rule counts

$$E_G[n_{A\to B\,C}|w] \;=\; \sum_{0\le i<j<k\le n} E_G[A_{i,k} \to B_{i,j} C_{j,k}|w]$$

The *expected fraction of parses of w in which $A_{i,k}$ rewrites as $B_{i,j}C_{j,k}$* is:

$$\begin{aligned} & E_G[A_{i,k} \to B_{i,j} C_{j,k}|w] \\ = \quad & \frac{\mathrm{P}(S \Rightarrow^* w_{1,i}\, A\, w_{k,n}) p(A \to B\, C) \mathrm{P}(B \Rightarrow^* w_{i,j}) \mathrm{P}(C \Rightarrow^* w_{j,k})}{\mathrm{P}_G(w)} \end{aligned}$$



Calculating $\mathrm{P}_G(S \Rightarrow^* w_{0,i}\, A\, w_{k,n})$

Known as "outside probabilities" (but if G contains unary productions, they can be greater than 1).

Recursion from *larger to smaller* substrings in w.

Base case: $\mathrm{P}(S \Rightarrow^* w_{0,0}\, S\, w_{n,n}) = 1$

Recursion: $\mathrm{P}(S \Rightarrow^* w_{0,j}\, C\, w_{k,n}) =$

$$\sum_{i=0}^{j-1} \sum_{\substack{A,B \in \mathcal{S} \\ A \to B\, C \in \mathcal{R}}} \mathrm{P}(S \Rightarrow^* w_{0,i}\, A\, w_{k,n}) p(A \to B\, C) \mathrm{P}(B \Rightarrow^* w_{i,j})$$

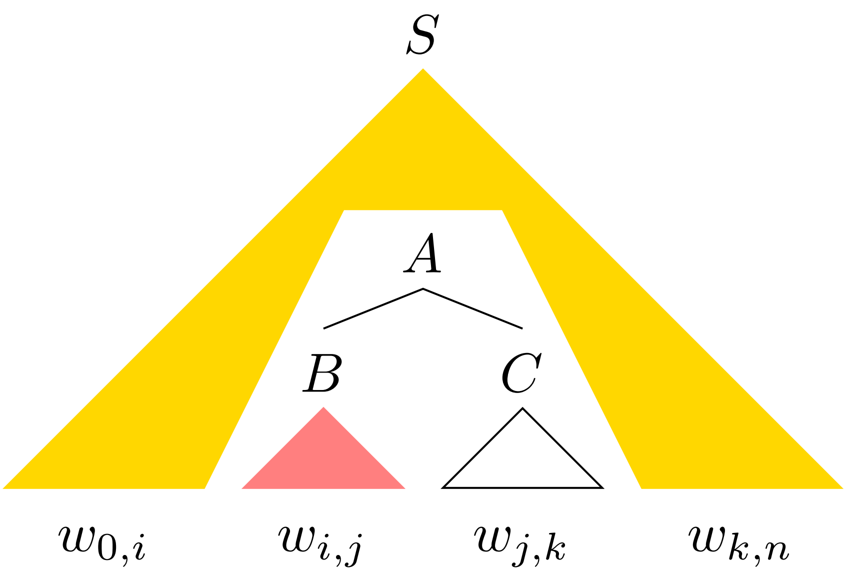
$$+ \sum_{l=k+1}^{n} \sum_{\substack{A,D \in \mathcal{S} \\ A \to C\, D \in \mathcal{R}}} \mathrm{P}(S \Rightarrow^* w_{0,j}\, A\, w_{l,n}) p(A \to C\, D) \mathrm{P}(D \Rightarrow^* w_{k,l})$$

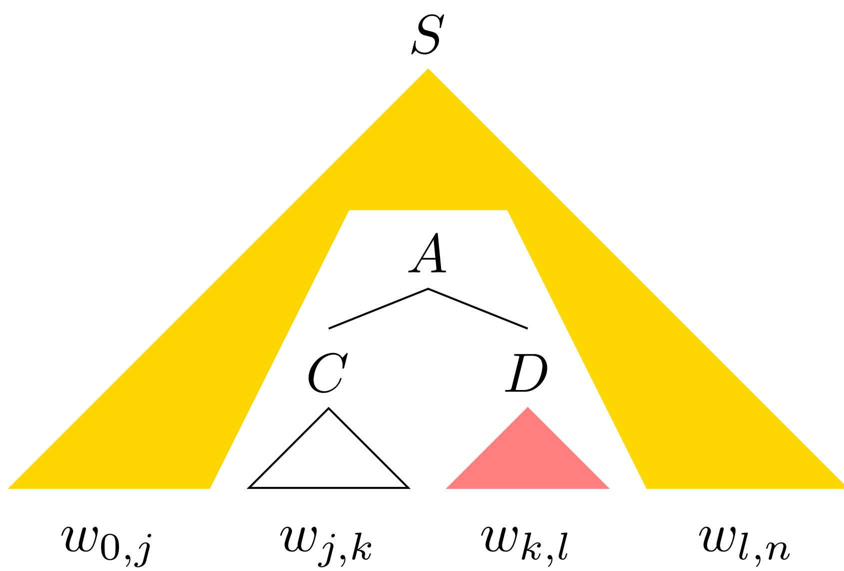
Recursion in $\mathrm{P}_G(S \Rightarrow^* w_{0,i}\, A\, w_{k,n})$

$$\mathrm{P}(S \Rightarrow^* w_{0,j}\, C\, w_{k,n}) =$$

$$\sum_{i=0}^{j-1} \sum_{\substack{A,B \in \mathcal{S} \\ A \to B\, C \in \mathcal{R}}} \mathrm{P}(S \Rightarrow^* w_{0,i}\, A\, w_{k,n}) p(A \to B\, C) \mathrm{P}(B \Rightarrow^* w_{i,j})$$

$$+ \sum_{l=k+1}^{n} \sum_{\substack{A,D \in \mathcal{S} \\ A \to C\, D \in \mathcal{R}}} \mathrm{P}(S \Rightarrow^* w_{0,j}\, A\, w_{l,n}) p(A \to C\, D) \mathrm{P}(D \Rightarrow^* w_{k,l})$$





The EM algorithm for PCFGs

Infer *hidden structure* by maximizing likelihood of *visible data*:

1. guess initial rule probabilities
2. repeat until convergence
   (a) parse a sample of sentences
   (b) weight each parse by its conditional probability
   (c) count rules used in each weighted parse, and estimate rule frequencies from these counts as before

EM optimizes the *marginal likelihood of the strings* $D = (w_1, \ldots, w_n)$

Each iteration is *guaranteed* not to decrease the likelihood of D, but EM can get trapped in local minima.

The *Inside-Outside algorithm* can produce the expected counts without enumerating all parses of D.

When used with PFSA, the Inside-Outside algorithm is called the *Forward-Backward algorithm* (Inside=Backward, Outside=Forward)

Example: The EM algorithm with a toy PCFG

Initial rule probs

rule	**prob**
...	...
VP $\longrightarrow$ V	0.2
VP $\longrightarrow$ V NP	0.2
VP $\longrightarrow$ NP V	0.2
VP $\longrightarrow$ V NP NP	0.2
VP $\longrightarrow$ NP NP V	0.2
...	...
Det $\longrightarrow$ the	0.1
N $\longrightarrow$ the	0.1
V $\longrightarrow$ the	0.1

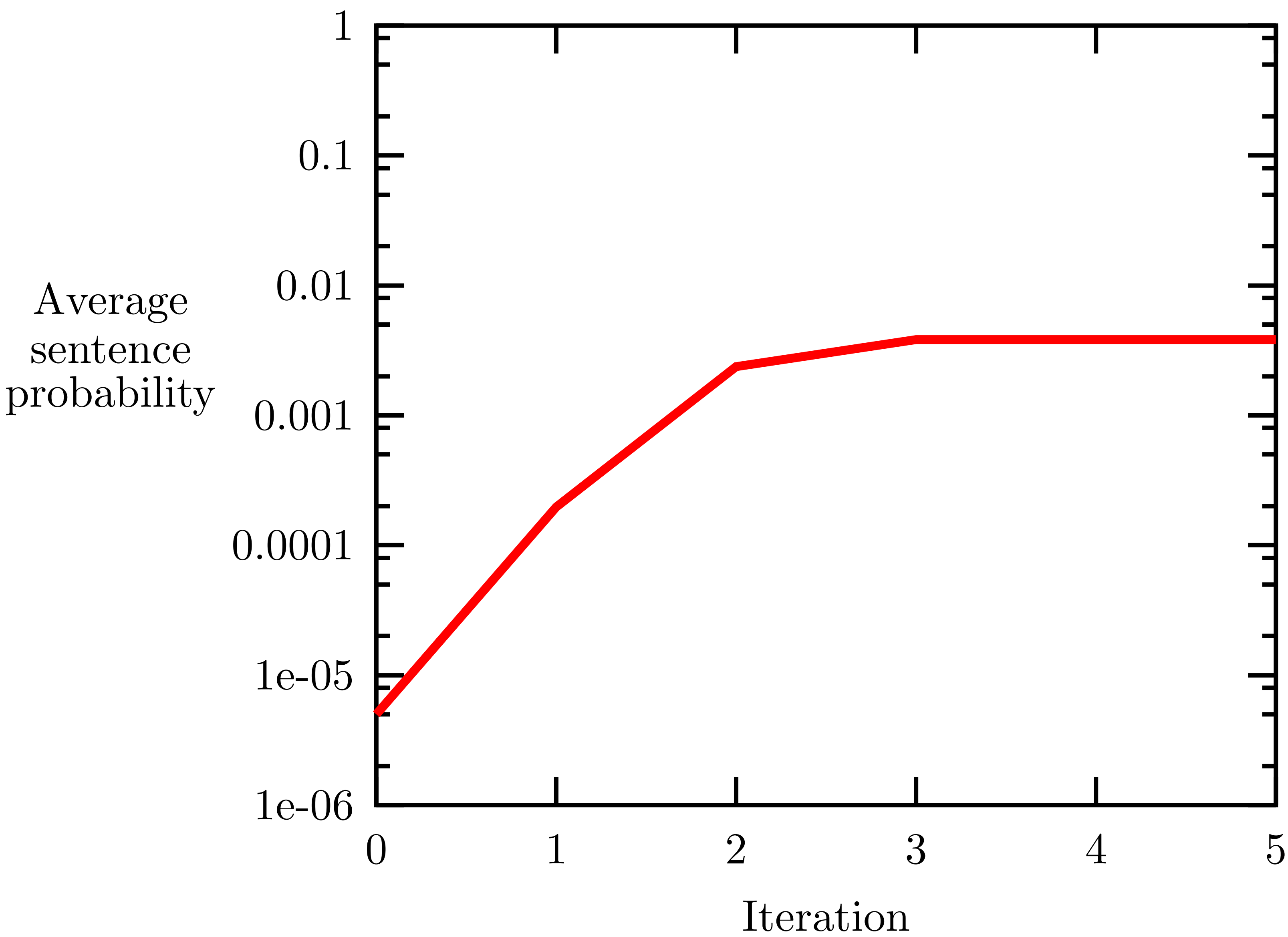
"English" input

the dog bites
the dog bites a man
a man gives the dog a bone
...

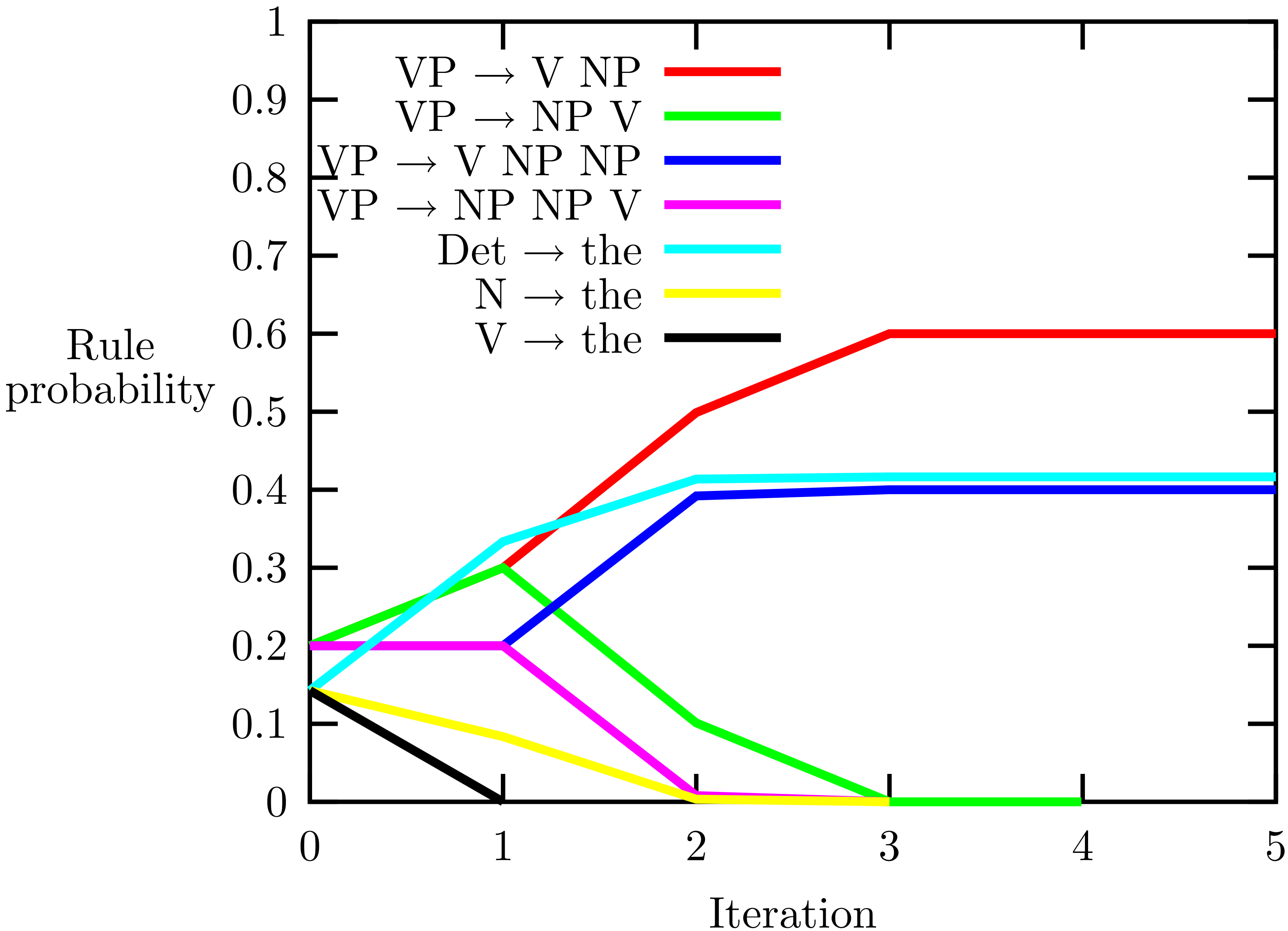
"pseudo-Japanese" input

the dog bites
the dog a man bites
a man the dog a bone gives
...

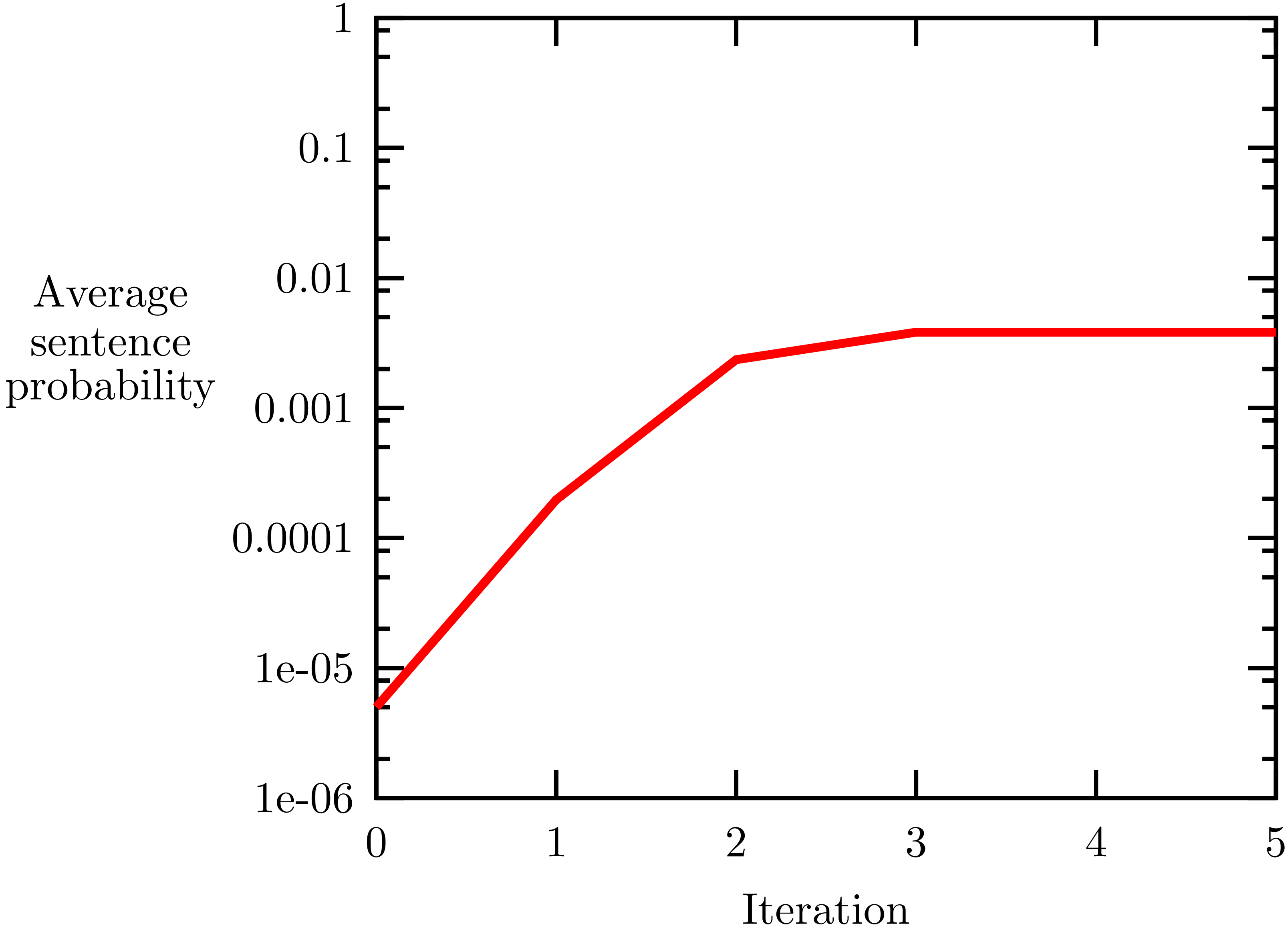
Probability of "English"



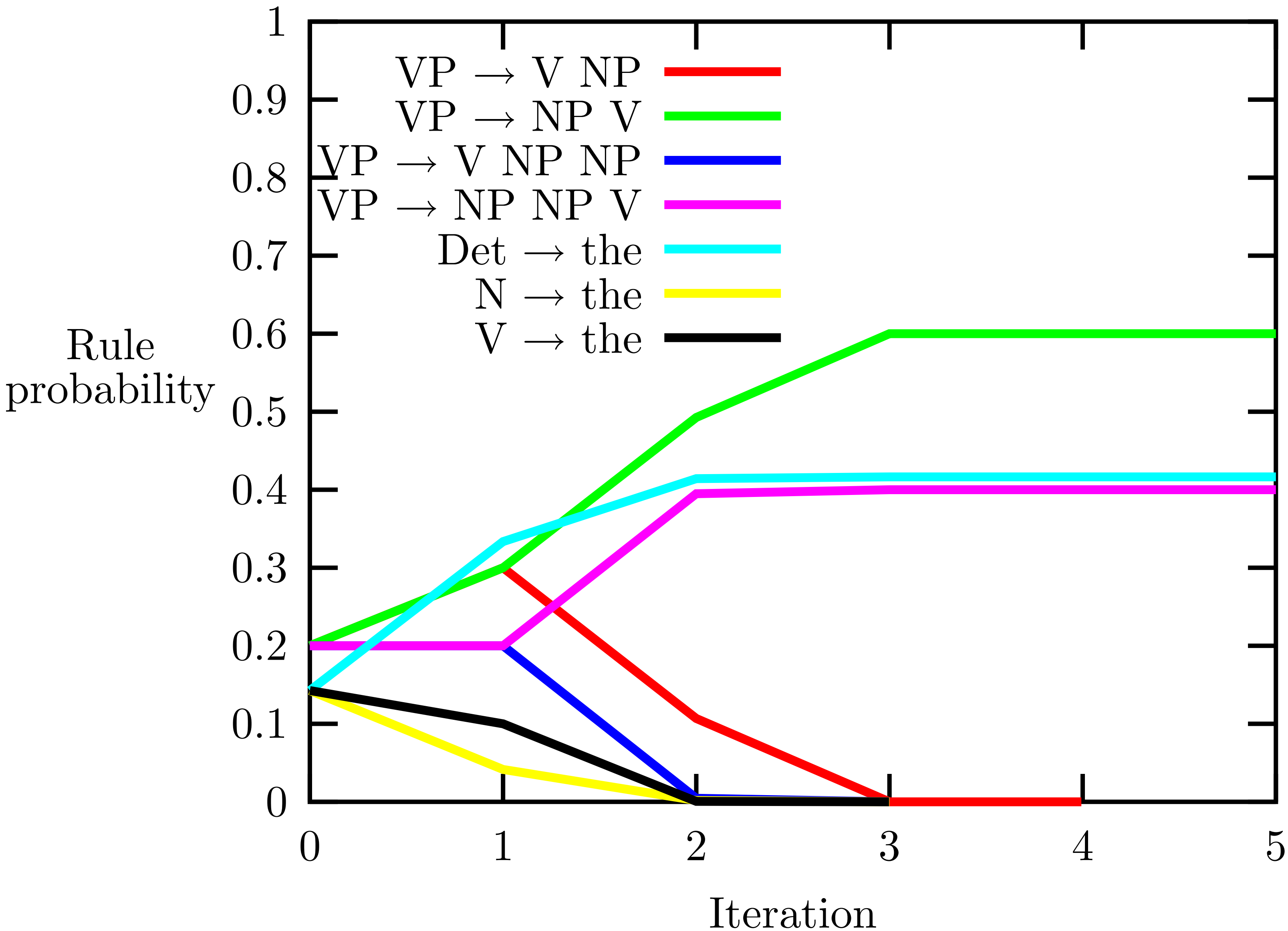
Rule probabilities from “English”



Probability of "Japanese"



Rule probabilities from “Japanese”

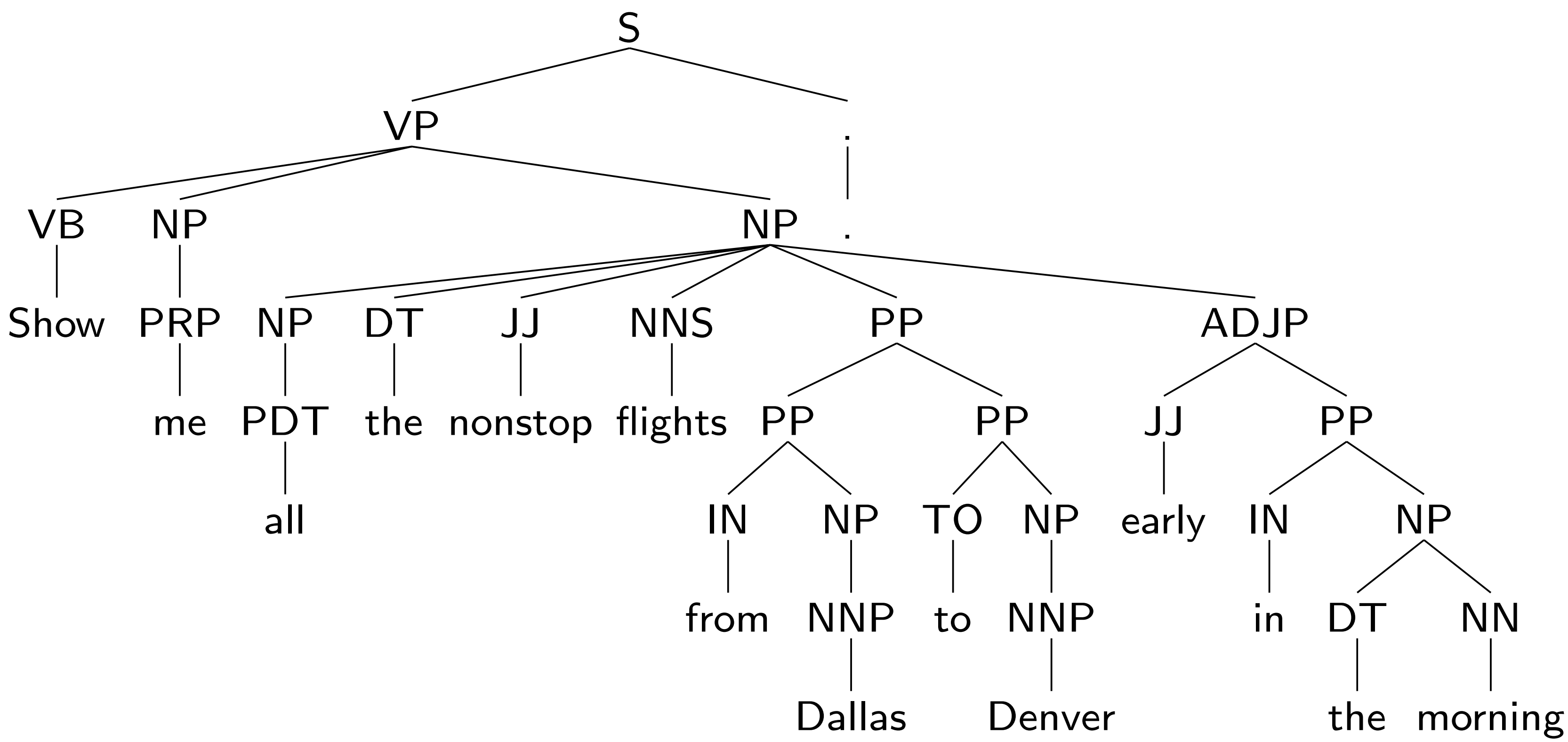


Learning in statistical paradigm

- The likelihood is a differentiable function of rule probabilities
  $\Rightarrow$ learning can involve small, incremental updates
- Learning new structure (rules) is hard, but ...
- Parameter estimation can approximate rule learning
  - start with "superset" grammar
  - estimate rule probabilities
  - discard low probability rules

Applying EM to real data

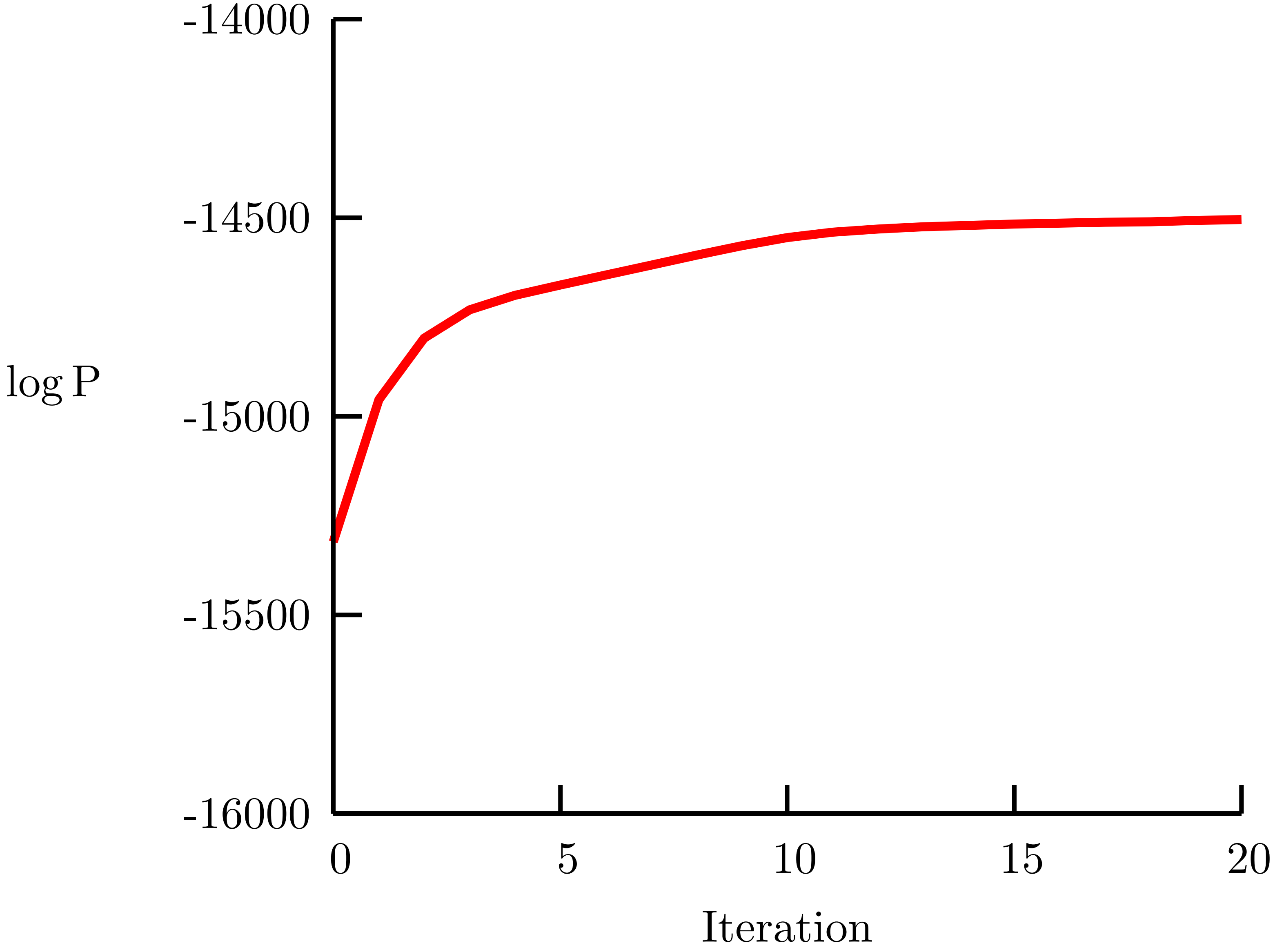
- ATIS treebank consists of 1,300 hand-constructed parse trees
- ignore the words (in this experiment)
- about 1,000 PCFG rules are needed to build these trees



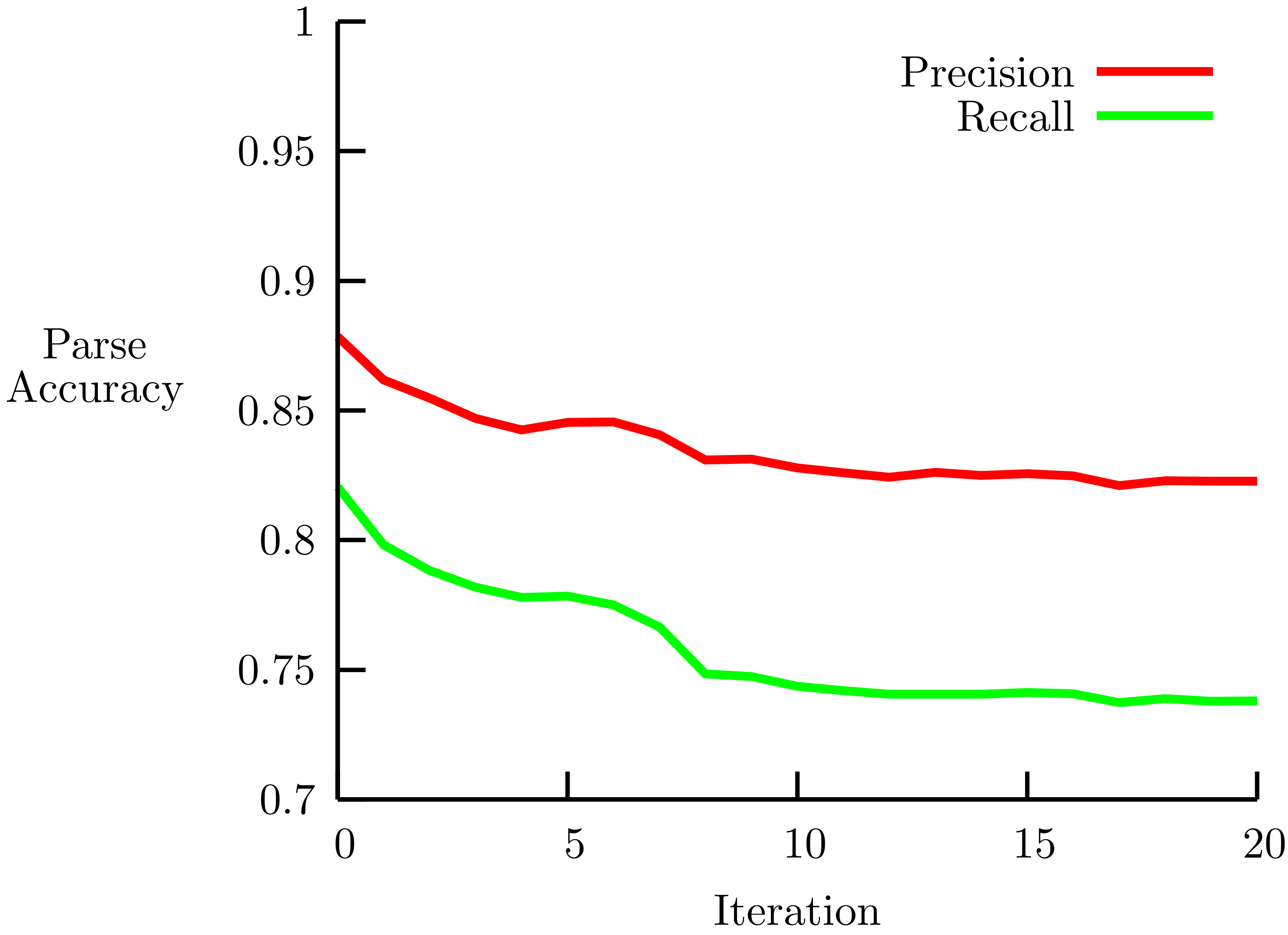
Experiments with EM

1. Extract productions from trees and estimate probabilities probabilities from trees to produce PCFG.
2. Initialize EM with the treebank grammar and MLE probabilities
3. Apply EM (to strings alone) to re-estimate production probabilities.
4. At each iteration:
   - Measure the likelihood of the training data and the quality of the parses produced by each grammar.
   - Test on training data (so poor performance is not due to overlearning).

Likelihood of training strings



Quality of ML parses



Why does EM do so poorly?

- *Wrong data:* grammar is a transduction between form and meaning $\Rightarrow$ learn from form/meaning pairs
  - exactly what contextual information is available to a language learner?
- *Wrong model:* PCFGs are poor models of syntax
- *Wrong objective function:* Maximum likelihood makes the sentences as likely as possible, but syntax isn't intended to predict sentences (Klein and Manning)
- How can information about the *marginal distribution of strings* $\mathrm{P}(w)$ provide information about the *conditional distribution of parses* ψ *given strings* $\mathrm{P}(\psi|w)$?
  - need additional *linking assumptions* about the relationship between parses and strings
- ... but no one really knows!

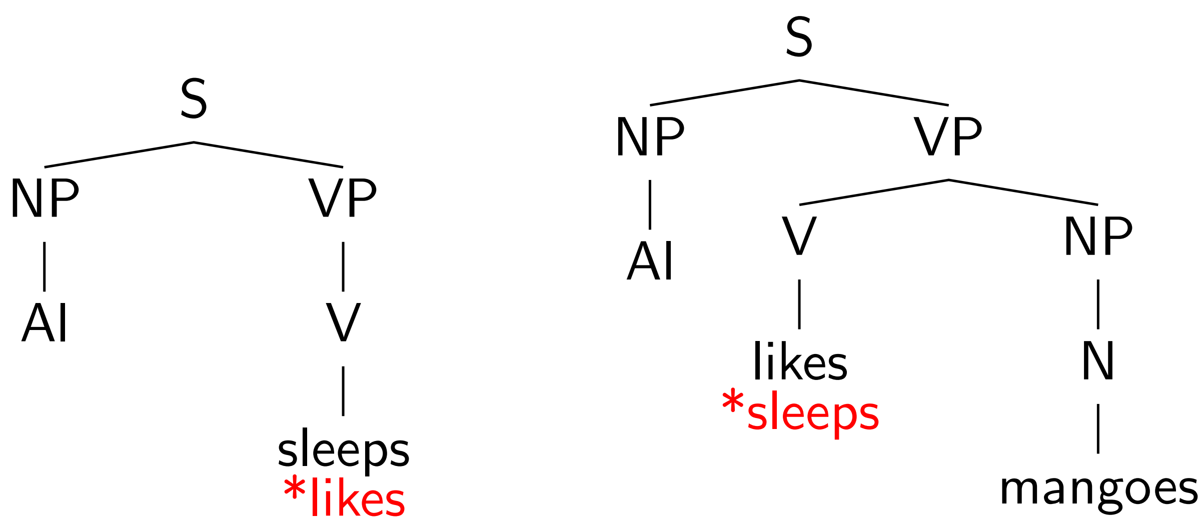
Topics

- Graphical models and Bayes networks
- (Hidden) Markov models
- (Probabilistic) context-free grammars
- (Probabilistic) finite-state machines
- Computation with PCFGs
- Estimation of PCFGs
- *Lexicalized and bi-lexicalized PCFGs*
- Non-local dependencies and log-linear models
- Stochastic unification-based grammars

Subcategorization

Grammars that merely relate categories miss a lot of important linguistic relationships.

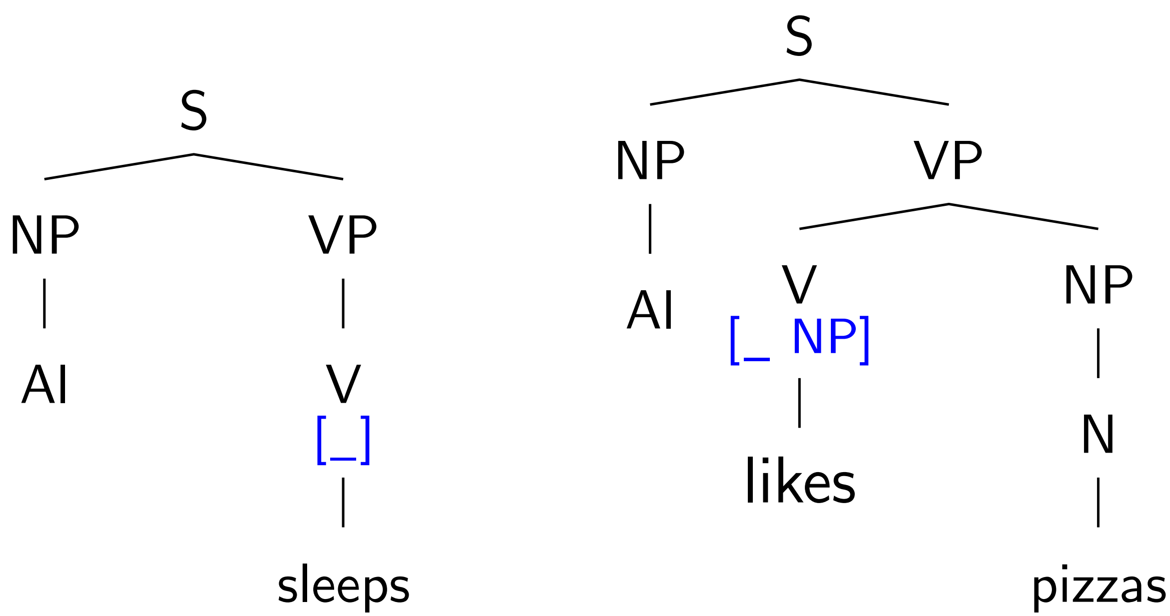
$R_3 = \{\mathsf{VP} \rightarrow \mathsf{V}, \mathsf{VP} \rightarrow \mathsf{V}\,\mathsf{NP}, \mathsf{V} \rightarrow \mathsf{sleeps}, \mathsf{V} \rightarrow \mathsf{likes}, \ldots\}$



Verbs and other *heads of phrases* subcategorize for the number and kind of *complement phrases* they can appear with.

CFG account of subcategorization

General idea: *Split the preterminal states* to encode subcategorization.

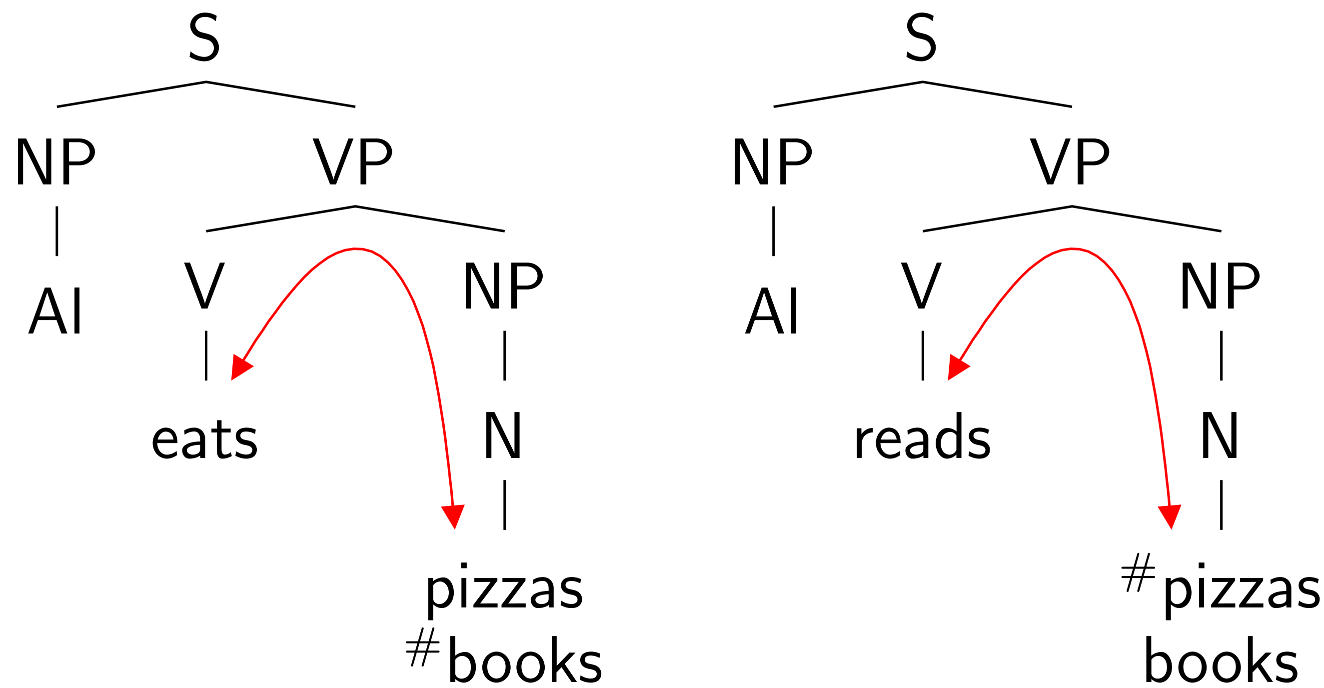


$$R_4 = \{\mathsf{VP} \rightarrow \underset{[\_]}{\mathsf{V}},\ \mathsf{VP} \rightarrow \underset{[\_\ \mathsf{NP}]}{\mathsf{V}}\ \mathsf{NP},\ \underset{[\_]}{\mathsf{V}} \rightarrow \mathsf{sleeps},\ \underset{[\_\ \mathsf{NP}]}{\mathsf{V}} \rightarrow \mathsf{likes}, \ldots\}$$

The "split preterminal states" restrict which contexts verbs can appear in.

Selectional preferences

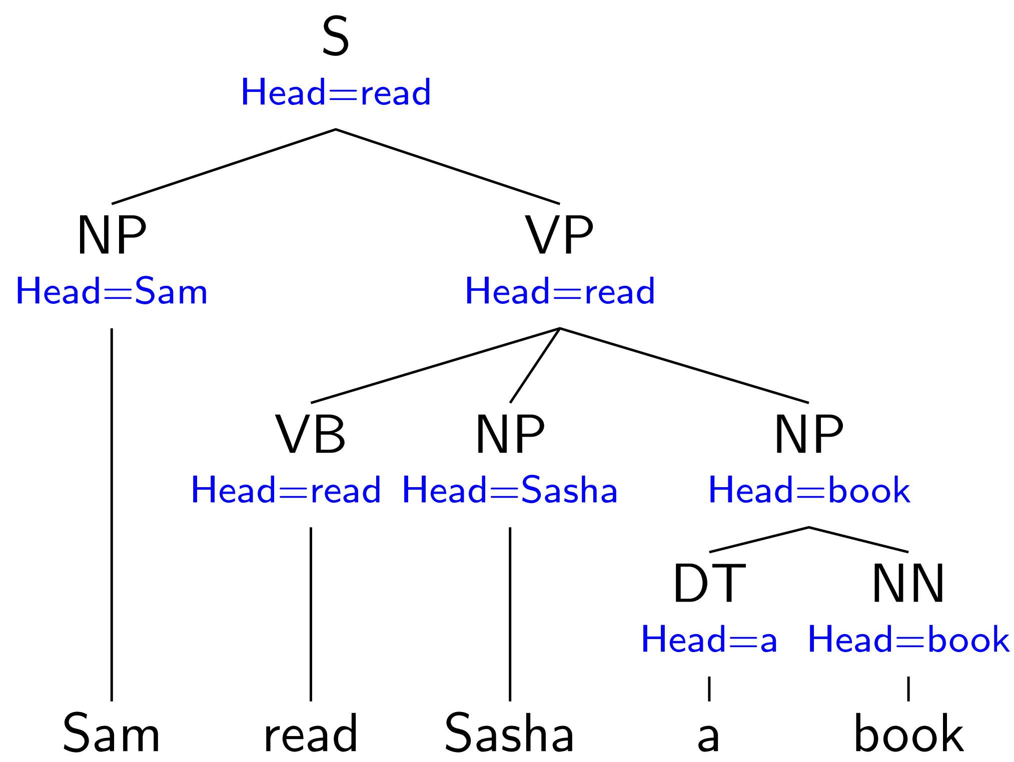
Head-to-head dependencies are an approximation to real-world knowledge.



But note that selectional preferences involve more than head-to-head dependencies

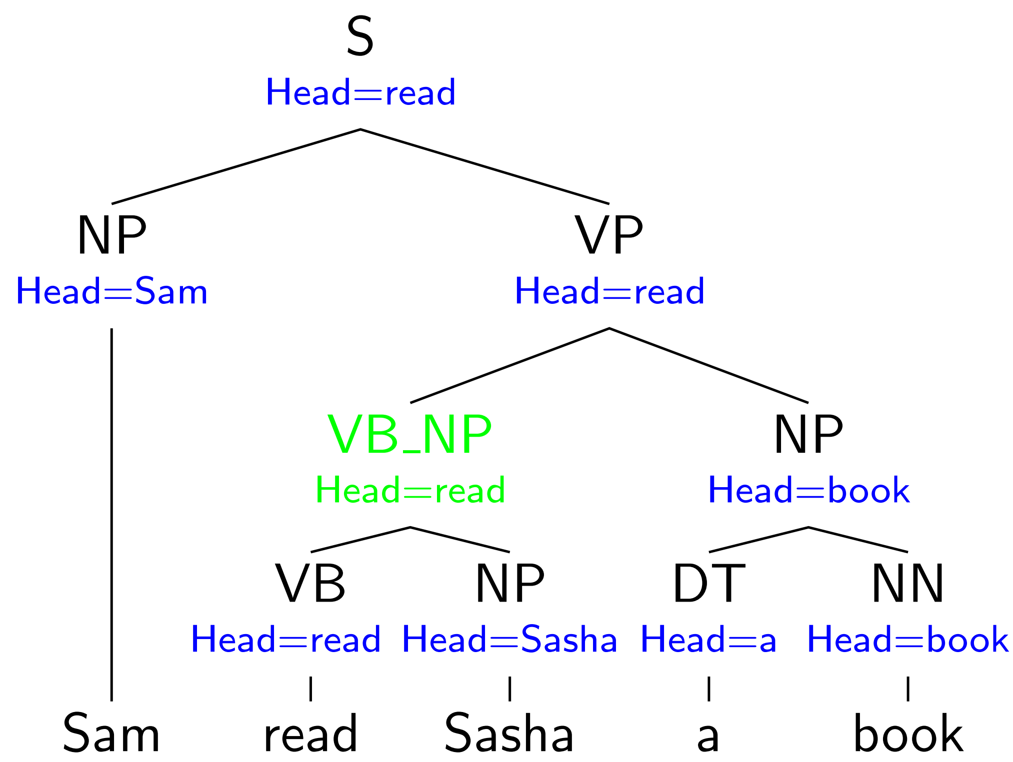
Al drives a ($^{\#}$toy model) car

Head to head dependencies



$$\underset{\text{Head}=\text{read}}{\text{VP}} \longrightarrow \underset{\text{Head}=\text{read}}{\text{VB}} \quad \underset{\text{Head}=\text{Sasha}}{\text{NP}} \quad \underset{\text{Head}=\text{book}}{\text{NP}}$$

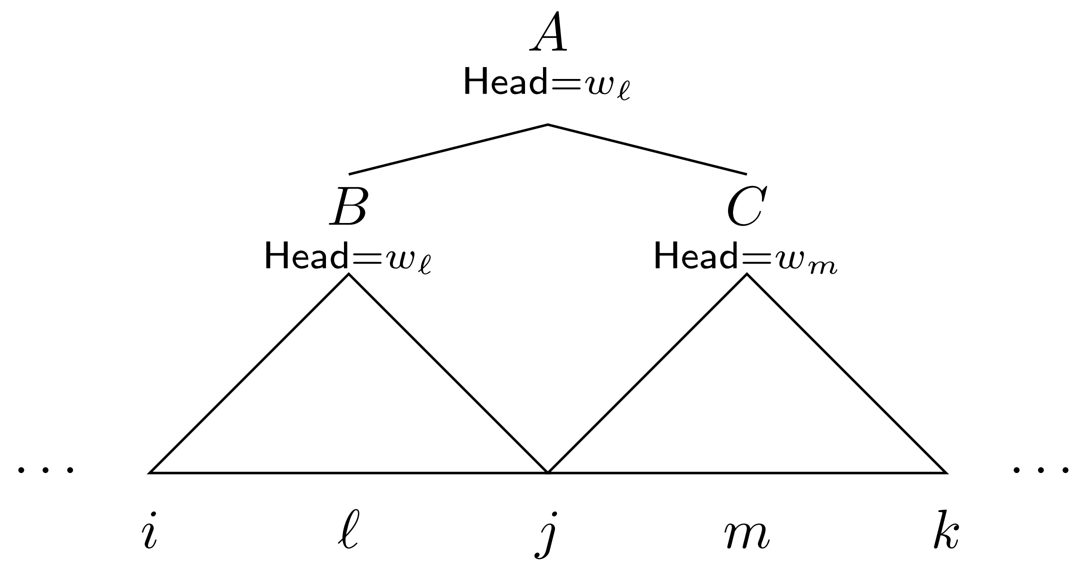
Binarization helps sparse data



$$\underset{\text{Head=read}}{\text{VP}} \longrightarrow \underset{\text{Head=read}}{\text{VB\_NP}} \quad \underset{\text{Head=book}}{\text{NP}}$$

$$\underset{\text{Head=read}}{\text{VB\_NP}} \longrightarrow \underset{\text{Head=read}}{\text{VB}} \quad \underset{\text{Head=Sasha}}{\text{NP}}$$

Bi-lexical CFG parsing takes n^5 time

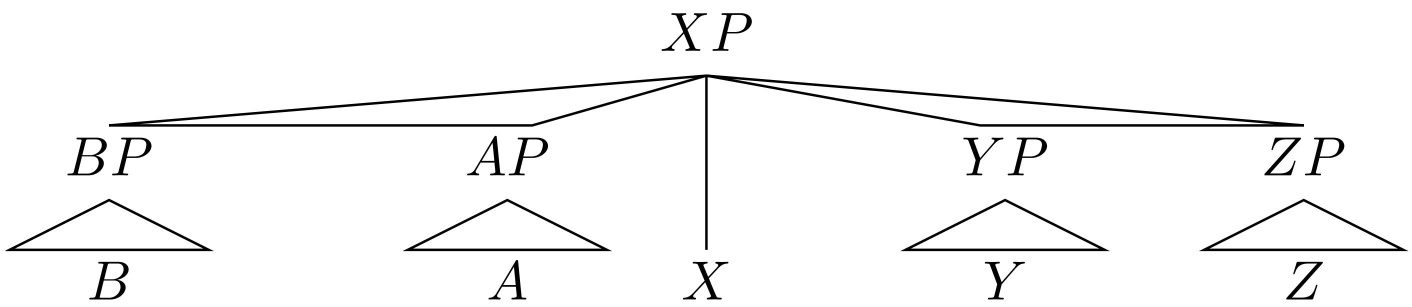


There are three string positions at the edges of constituents, plus two for the locations of the heads

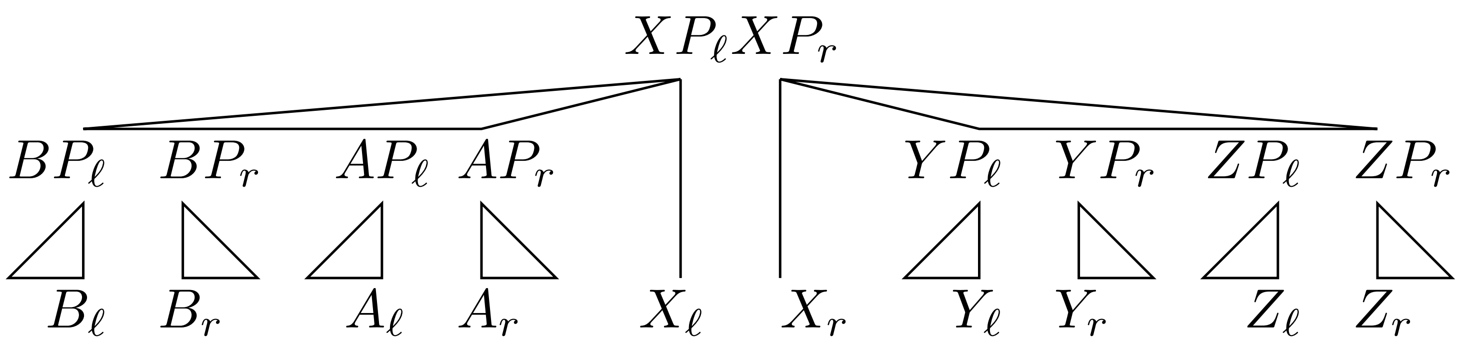
- in the worst case, bilexical parsing takes $|n|^5$ time
- the worst case arises when exhaustive parsing

Eisner and Satta's idea: transform the grammar so that the *heads are at the constituent edges* (alternatively, approximate the CFG by a *dependency grammar*)

Eisner and Satta's bilexical parsing model



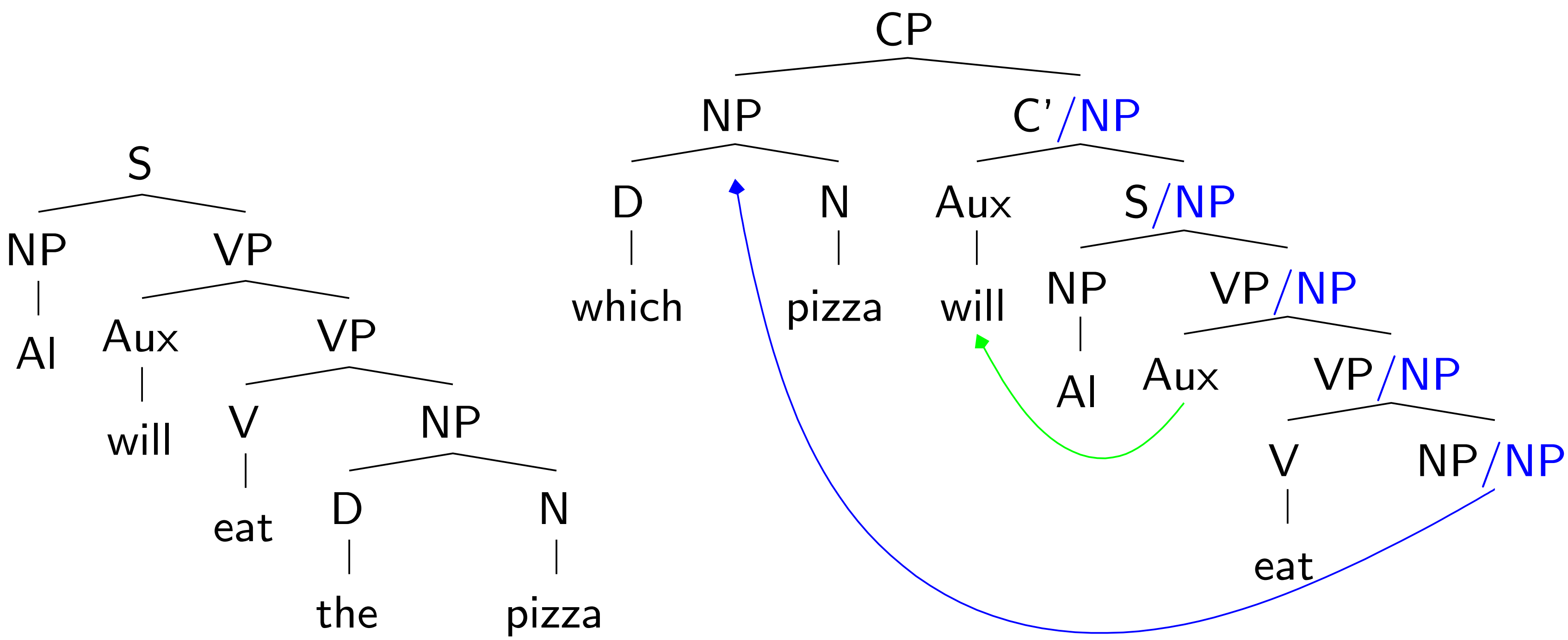
Split each node (including each word) into a left and a right half



Right factor the left halves and left factor the right halves

Synchronize left and right halves if needed by *splitting the nonterminal states*

Nonlocal “movement” dependencies

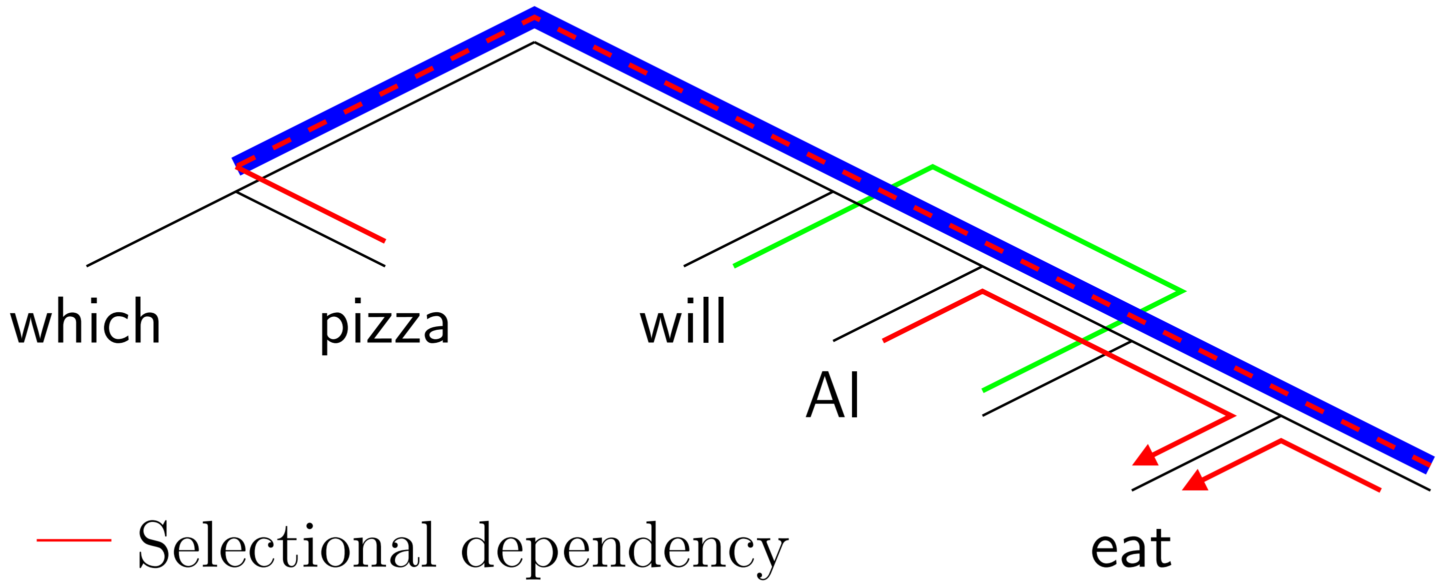


Subcategorization and selectional preferences are *preserved* under movement.

Movement can be encoded using *recursive* nonterminals (unification grammars).

Structured nonterminals

Structured nonterminals provide *communication channels* that pass information around the tree.



— Selectional dependency

— Verb movement dependency

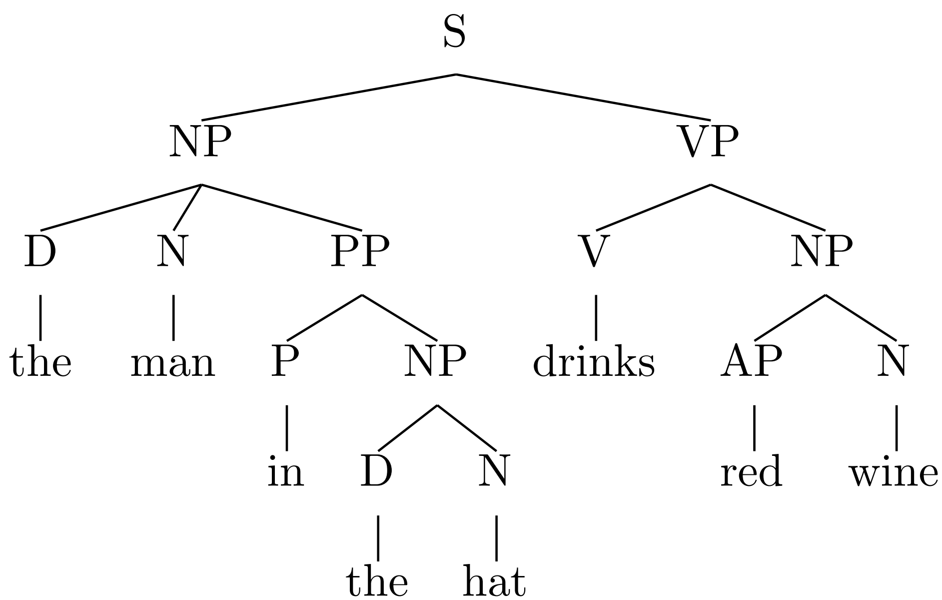
— WH movement dependency

Modern statistical parsers pass around 7 different features through the tree, and condition productions on them

Topics

- Graphical models and Bayes networks
- (Hidden) Markov models
- (Probabilistic) context-free grammars
- (Probabilistic) finite-state machines
- Computation with PCFGs
- Estimation of PCFGs
- Lexicalized and bi-lexicalized PCFGs
- *Non-local dependencies and log-linear models*
- Stochastic unification-based grammars

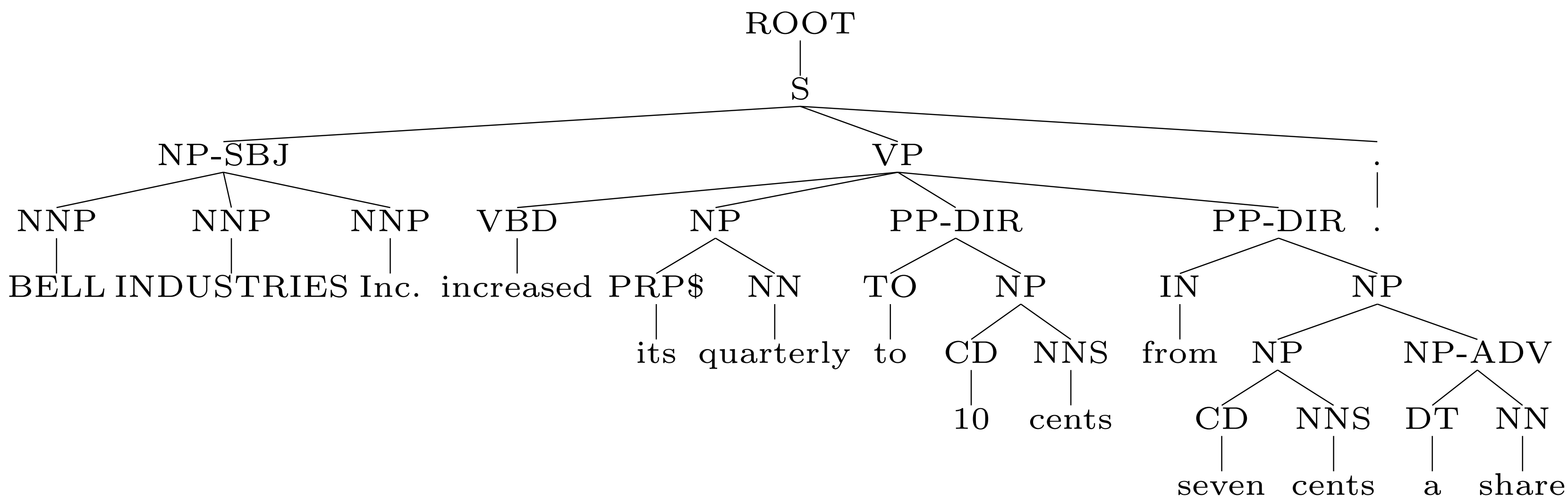
Probabilistic Context Free Grammars



$$
\begin{aligned}
P(t) \quad = \quad & P(S \rightarrow NP\ VP) \times \\
& P(NP \rightarrow D\ N\ PP) \times \\
& P(D \rightarrow \text{the}) \times \\
& P(N \rightarrow \text{man}) \times \\
& \ldots
\end{aligned}
$$

- Rules are associated with *probabilities*
- Tree probability is the *product of rule probabilities*
- *Most probable tree* is "best guess" at correct syntactic structure

Treebank corpora



- The Penn treebank contains hand-annotated parse trees for $\sim 50,000$ sentences
- Treebanks also exist for the Brown corpus, the Switchboard corpus (spontaneous telephone conversations) and Chinese and Arabic corpora

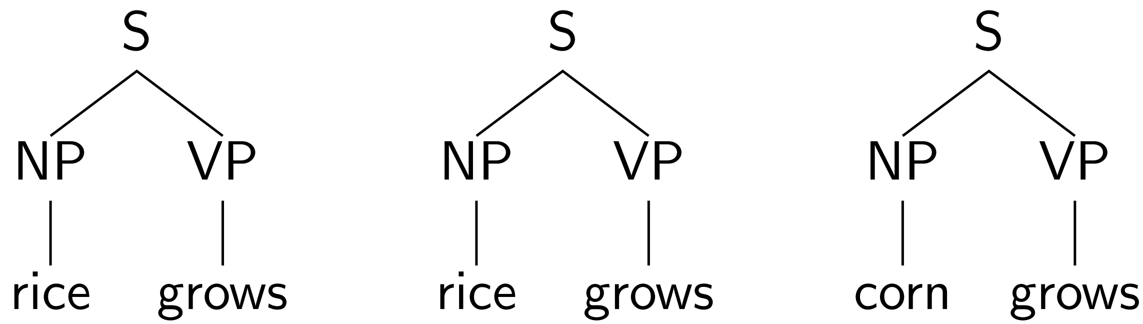
Estimating a grammar from a treebank

- *Maximum likelihood principle*: Choose the grammar and rule probabilities that make the trees in the corpus *as likely as possible*
  - read the rules off the trees
  - for PCFGs, set rule probabilities to the *relative frequency* of each rule in the treebank

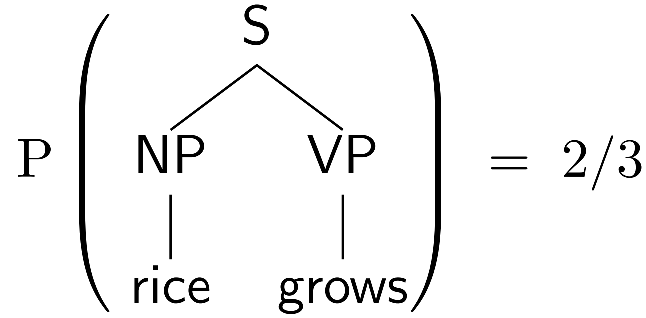
$$\mathrm{P}(\mathrm{VP} \rightarrow \mathrm{V\ NP}) \;=\; \frac{\text{Number of times VP} \rightarrow \text{V NP occurs}}{\text{Number of times VP occurs}}$$

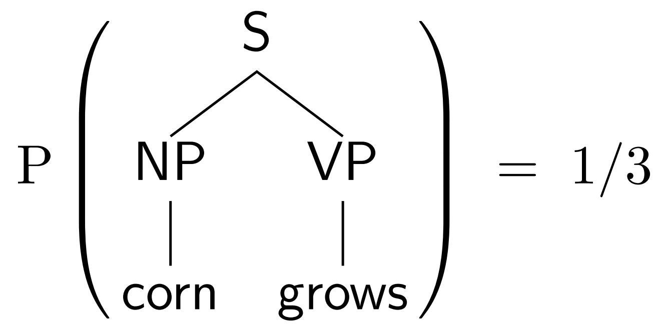
- *If the language is generated by a PCFG and the treebank trees are its derivation trees, the estimated grammar converges to the true grammar.*

Estimating PCFGs from visible data

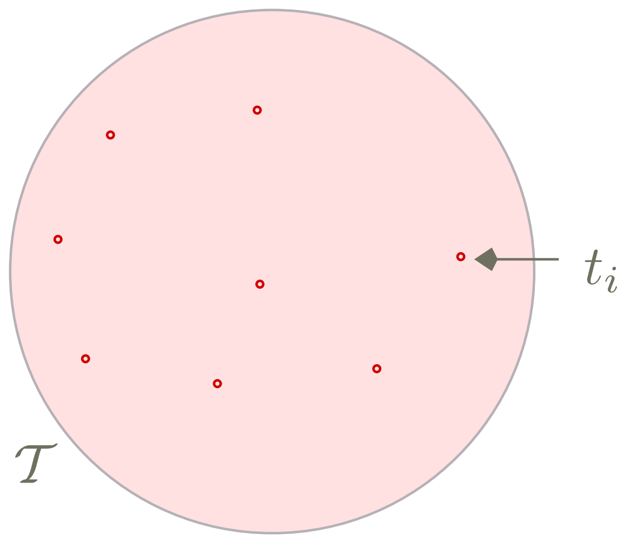


Rule	Count	Rel Freq
S $\longrightarrow$ NP VP	3	1
NP $\longrightarrow$ rice	2	2/3
NP $\longrightarrow$ corn	1	1/3
VP $\longrightarrow$ grows	3	1



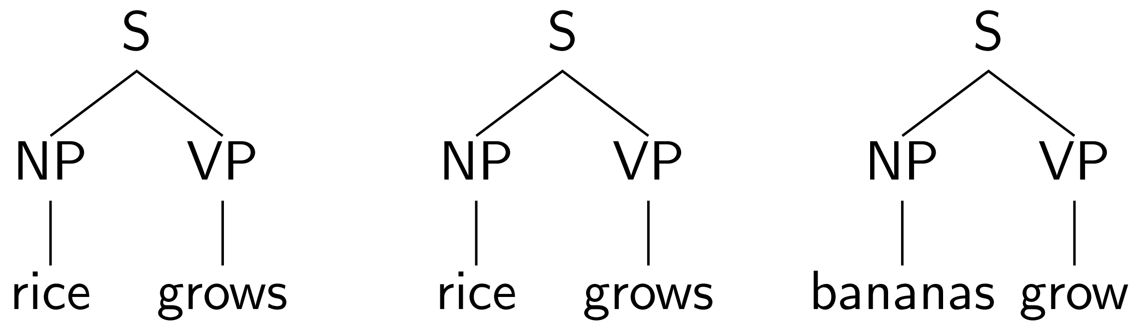


Why is the PCFG MLE so easy to compute?

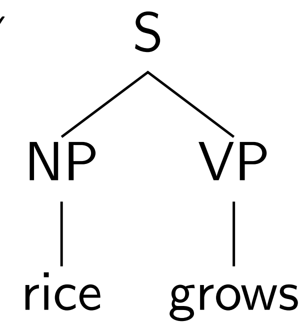


- Visible training data $D = (t_1, \ldots, t_n)$, where t_i is a parse tree
- The MLE is $\widehat{w} = \arg\max_w \prod_{i=1}^{n} \mathrm{P}_w(t_i)$, where w are production probabilities
- It is easy to compute because PCFGs are *always normalized*, i.e., $Z = \sum_{t \in \mathcal{T}} \prod_r w(r)^{f_r(t)} = 1$, where:
  - $f_r(t)$ is number of times r is used in derivation of t and
  - $\mathcal{T}$ is the set of all trees generated by the grammar

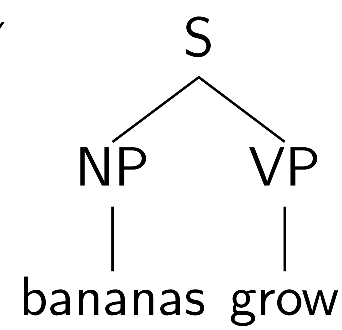
Non-local constraints and PCFG MLE



Rule	**Count**	**Rel Freq**
S $\rightarrow$ NP VP	3	1
NP $\rightarrow$ rice	2	2/3
NP $\rightarrow$ bananas	1	1/3
VP $\rightarrow$ grows	2	2/3
VP $\rightarrow$ grow	1	1/3

P (


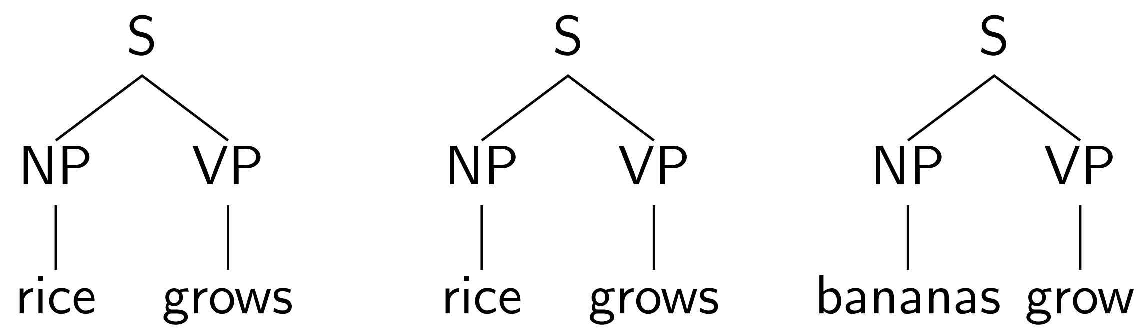
) $= 4/9$

P (


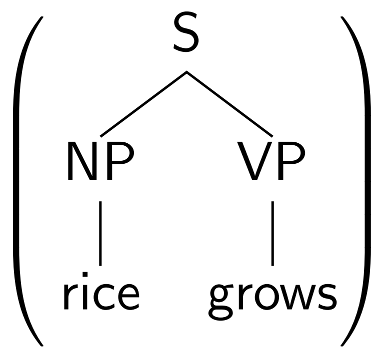
) $= 1/9$

partition function $Z = 5/9$

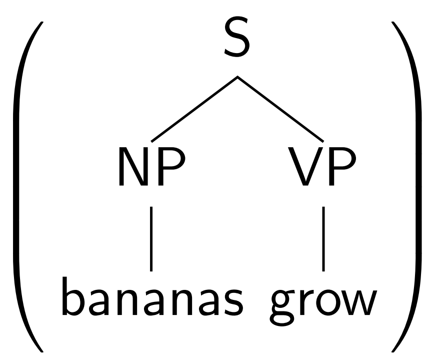
Dividing by partition function Z



Rule	Count	Rel Freq
S $\longrightarrow$ NP VP	3	1
NP $\longrightarrow$ rice	2	$2/3$
NP $\longrightarrow$ bananas	1	$1/3$
VP $\longrightarrow$ grows	2	$2/3$
VP $\longrightarrow$ grow	1	$1/3$

P 

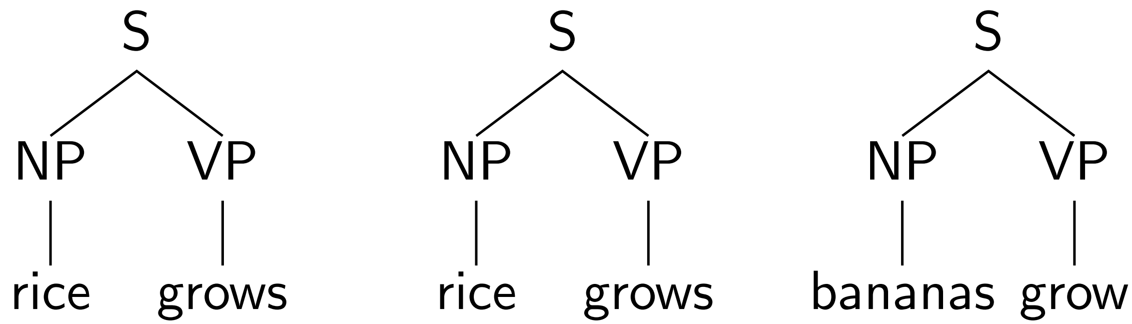
$= \quad 4/9 \quad 4/5$

P 

$= \quad 1/9 \quad 1/5$

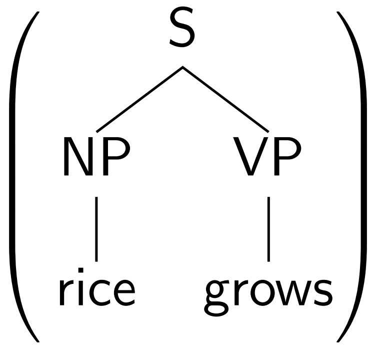
$Z = 5/9$

Other values do better!

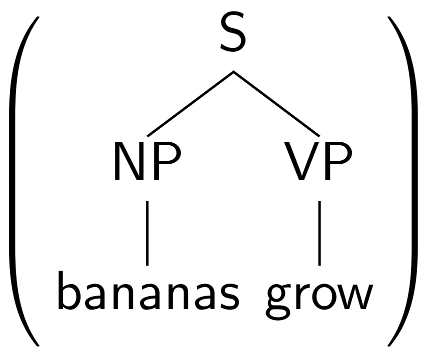


Rule	Count	Rel Freq
S $\rightarrow$ NP VP	3	1
NP $\rightarrow$ rice	2	2/3
NP $\rightarrow$ bananas	1	1/3
VP $\rightarrow$ grows	2	1/2
VP $\rightarrow$ grow	1	1/2

(Abney 1997)

P 

$= \quad 2/6 \quad 2/3$

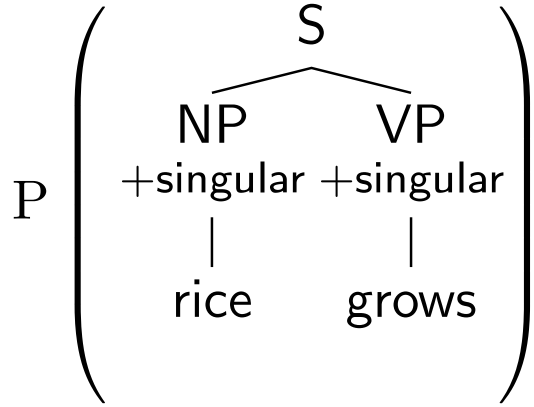
P 

$= \quad 1/6 \quad 1/3$

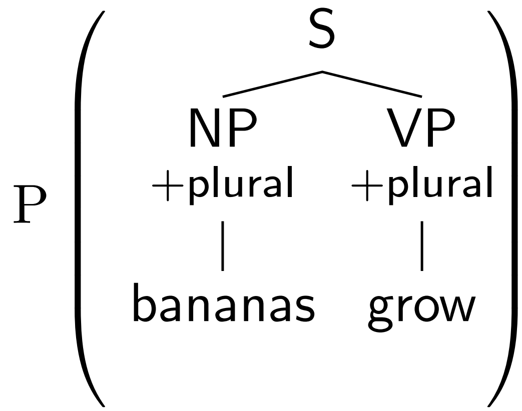
$Z = 3/6$

Make dependencies local – GPSG-style

rule	count	rel freq
$\mathsf{S} \longrightarrow \underset{+\mathsf{singular}}{\mathsf{NP}}\ \underset{+\mathsf{singular}}{\mathsf{VP}}$	2	2/3
$\mathsf{S} \longrightarrow \underset{+\mathsf{plural}}{\mathsf{NP}}\ \underset{+\mathsf{plural}}{\mathsf{VP}}$	1	1/3
$\underset{+\mathsf{singular}}{\mathsf{NP}} \longrightarrow \mathsf{rice}$	2	1
$\underset{+\mathsf{plural}}{\mathsf{NP}} \longrightarrow \mathsf{bananas}$	1	1
$\underset{+\mathsf{singular}}{\mathsf{VP}} \longrightarrow \mathsf{grows}$	2	1
$\underset{+\mathsf{plural}}{\mathsf{VP}} \longrightarrow \mathsf{grow}$	1	1

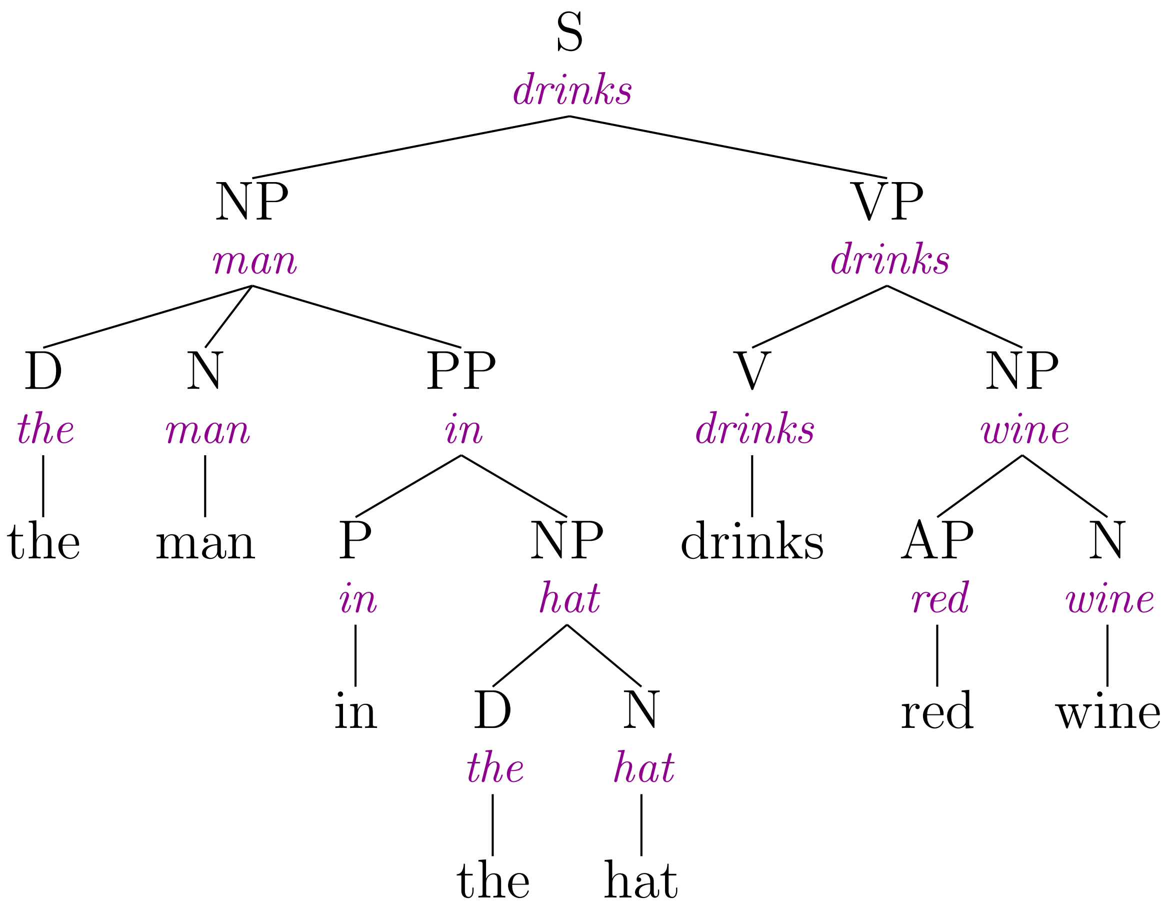


$= 2/3$



$= 1/3$

"Head to head" dependencies



Rules:

$$\underset{drinks}{\text{S}} \longrightarrow \underset{man}{\text{NP}} \quad \underset{drinks}{\text{VP}}$$

$$\underset{drinks}{\text{VP}} \longrightarrow \underset{drinks}{\text{V}} \quad \underset{wine}{\text{NP}}$$

$$\underset{wine}{\text{NP}} \longrightarrow \underset{red}{\text{AP}} \quad \underset{wine}{\text{N}}$$

$\ldots$

- *Lexicalization* captures syntactic and semantic dependencies
- Lexicalized structural preferences may be most important

Summary so far

- Maximum likelihood is a good way of estimating a grammar
- Maximum likelihood estimation of a PCFG from a treebank is easy *if the trees are accurate*
- But real language has many more dependencies than treebank grammar describes

⇒ relative frequency estimator not MLE

  – Make non-local dependencies local by splitting categories

  ⇒ Astronomical number of possible categories

- Or find some way of dealing with non-local dependencies . . .

Exponential models

- Rules are not independent $\Rightarrow Z \neq 1$, relative frequency estimator not MLE
- *Exponential models* permit dependencies between features
  - Universe $\mathcal{T}$ (set of all possible parse trees)
  - Features $f = (f_1, \ldots, f_m)$ ($f_j(t)$ = value of j feature on $t \in \mathcal{T}$)
  - Feature weights $w = (w_1, \ldots, w_m)$

$$
\begin{aligned}
\mathrm{P}(t) &= \frac{1}{Z} \exp w \cdot f(t) = \frac{1}{Z} \exp \sum_{j=1}^{m} w_j f_j(t) \\
Z &= \sum_{t' \in \mathcal{T}} \exp w \cdot f(t') = \sum_{t' \in \mathcal{T}} \exp \sum_{j=1}^{m} w_j f_j(t')
\end{aligned}
$$

Hint: Think of $\exp w \cdot f(t)$ as unnormalized probability of t

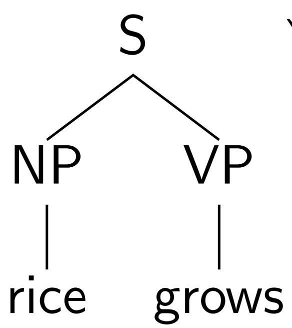
PCFGs are exponential models

$\mathcal{T}$ = set of all trees generated by PCFG G

$f_j(t)$ = number of times the jth rule is used in $t \in \mathcal{T}$

$p(r_j)$ = probability of jth rule in G

Set weight $w_j = \log p(r_j)$

$f\Bigg($ 

$\Bigg)$ $= [\underbrace{1}_{\text{S}\rightarrow\text{NP VP}}, \underbrace{1}_{\text{NP}\rightarrow\text{rice}}, \underbrace{0}_{\text{NP}\rightarrow\text{bananas}}, \underbrace{1}_{\text{VP}\rightarrow\text{grows}}, \underbrace{0}_{\text{VP}\rightarrow\text{grow}}]$

$$\mathrm{P}_w(t) = \prod_{j=1}^{m} p(r_j)^{f_j(t)} = \prod_{j=1}^{m} (\exp w_j)^{f_j(t)} = \exp(w \cdot f(t))$$

So *a PCFG is just a special kind of exponential model* with $Z = 1$.

Advantages of exponential models

- Exponential models are very flexible ...
- Features f can be *any function of parses* ...
  - whether a particular structure occurs in a parse
  - conjunctions of prosodic and syntactic structure
- Parses t need not be trees, but *can be anything at all*
  - Feature structures (LFG, HPSG), Minimalist derivations
- Exponential models are the same as (related to?) other popular models
  - Harmony theory (and hence optimality theory)
  - Maxent models
    * A Maximum Entropy model is one which has as much entropy as possible in the set of models whose expected feature counts equal the true feature counts of the data
    * the same model as the maximum likelihood model with the same features

MLE of exponential models and expectations

$$
\begin{aligned}
D &= (t_1, \ldots, t_n) \qquad \text{(treebank trees)} \\
\mathrm{P}_w(t) &= \frac{1}{Z} \exp w \cdot f(t) \\
Z &= \sum_{t \in \mathcal{T}} \exp w \cdot f(t) \qquad \text{(partition function)} \\
\ell_D(w) &= \sum_{i=1}^{n} \log \mathrm{P}_w(t_i) \;=\; \sum_{i=1}^{n} \log \frac{1}{Z} \exp w \cdot f(t_i) \\
&= w \cdot \left(\sum_{i=1}^{n} f(t_i) \right) - n \log Z \\
\frac{\partial \ell_D(w)}{\partial w_j} &= \sum_{i=1}^{n} f_j(t_i) - \frac{n}{Z} \sum_{t \in \mathcal{T}} f_j(t) \exp w \cdot f(t) \\
&= \sum_{i=1}^{n} f_j(t_i) - n \mathrm{E}_w[f_j], \text{where } \mathrm{E}_w[f] \;=\; \sum_{t \in \mathcal{T}} f(t) \mathrm{P}_w(t)
\end{aligned}
$$

Modeling dependencies

- It's usually difficult to design a PCFG model that captures a particular set of dependencies
  - probability of the tree must be broken down into a product of independent *conditional probability distributions* (c.f., Bayes nets)
  - non-local dependencies must be expressed in terms of GPSG-style feature passing
- It's easy to make exponential models sensitive to new dependencies
  - add a new feature functions to existing feature functions
  - estimation is a harder computational problem (see below)
  - conditional estimation ⇒ feature dependencies don't matter
  - *figuring out what the right dependencies are is hard, but incorporating them into an exponential model is easy*

Finding MLE of exponential models is hard

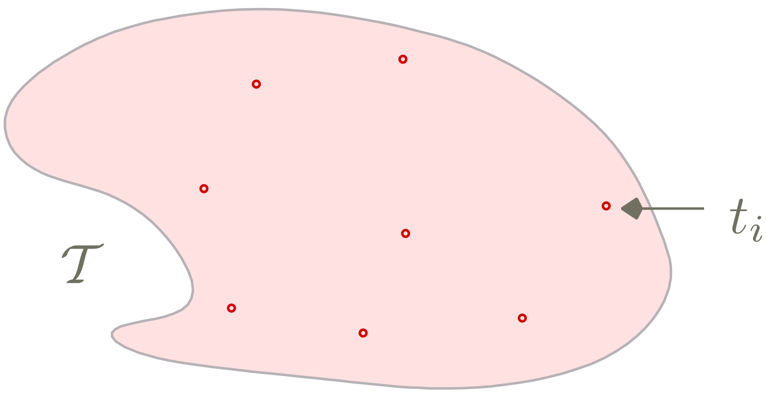
- An exponential model associates features $f(t) = (f_1(t), \ldots, f_m(t))$ with weights $w = (w_1, \ldots, w_m)$

$$\begin{aligned} \mathrm{P}(t) &= \frac{1}{Z} \exp w \cdot f(t) \\ Z &= \sum_{t' \in \mathcal{T}} \exp w \cdot f(t') \end{aligned}$$

- Given treebank $(t_1, \ldots, t_n)$, MLE chooses w to maximize $\mathrm{P}(t_1) \times \ldots \times \mathrm{P}(t_n)$, i.e., *make the treebank as likely as possible*
- Computing $\mathrm{P}(t)$ requires the partition function Z
- Computing Z requires a sum over all parses $\mathcal{T}$ for *all sentences*

$\Rightarrow$ *computing MLE of an exponential parsing model seems very hard*

ML estimation for exponential models



$$
\begin{aligned}
D &= (t_1, \ldots, t_n) \\
L_D(w) &= \prod_{i=1}^{n} \mathrm{P}_w(t_i) \\
\widehat{w} &= \operatorname*{arg\,max}_w L_D(w) \\
&= \operatorname*{arg\,max}_w \prod_{i=1}^{n} \mathrm{P}_w(t_i)
\end{aligned}
$$

$$
\mathrm{P}_w(t) = \frac{V_w(t)}{Z_w}, \quad V_w(t) = \exp \sum_j w_j f_j(t), \quad Z_w = \sum_{t' \in \mathcal{T}} V_w(t')
$$

- $\mathcal{T}$ is set of all possible parses for all possible strings
- For a PCFG, $\widehat{w}$ is easy to calculate, but ...
- in general $\partial L_D / \partial w_j$ and Z_w are *intractable analytically and numerically*
- Abney (1997) suggests a Monte-Carlo calculation method

Conditional ML estimation

- Conditional ML estimation chooses feature weights to maximize $\mathrm{P}_w(t_1|s_1) \times \ldots \times \mathrm{P}_w(t_n|s_n)$, where s_i is string for t_i
  - choose feature weights to make t_i most likely relative to parses $\mathcal{T}(s_i)$ for s_i

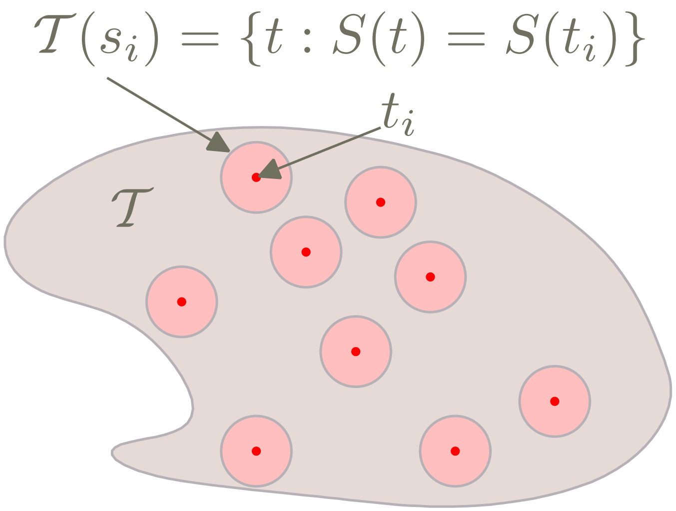
  $\Rightarrow$ CMLE *doesn't involve parses of other sentences*

$$
\begin{aligned}
\mathrm{P}_w(t|s) &= \frac{1}{Z_w(s)} \exp w \cdot f(t) \\
Z_w(s) &= \sum_{t' \in \mathcal{T}(s)} \exp w \cdot f(t')
\end{aligned}
$$

- $\mathcal{T}(s)$ is set of all parses for string s
- CMLE "only" involves repeatedly parsing training data
- With "wrong" models, CMLE often produces a more accurate parser than joint MLE

Conditional estimation

The *conditional likelihood* of w is the *conditional probability* of the *hidden part* (syntactic structure) t given its *visible part* (yield or terminal string) $s = S(t)$



$$
\begin{aligned}
\widehat{w}' &= \arg\max_{w} L'_D(w) \\
L'_D(w) &= \prod_{i=1}^{n} \mathrm{P}_w(t_i|s_i) \\
\mathrm{P}_w(t|s) &= \frac{V_w(t)}{Z_w(s)}
\end{aligned}
$$

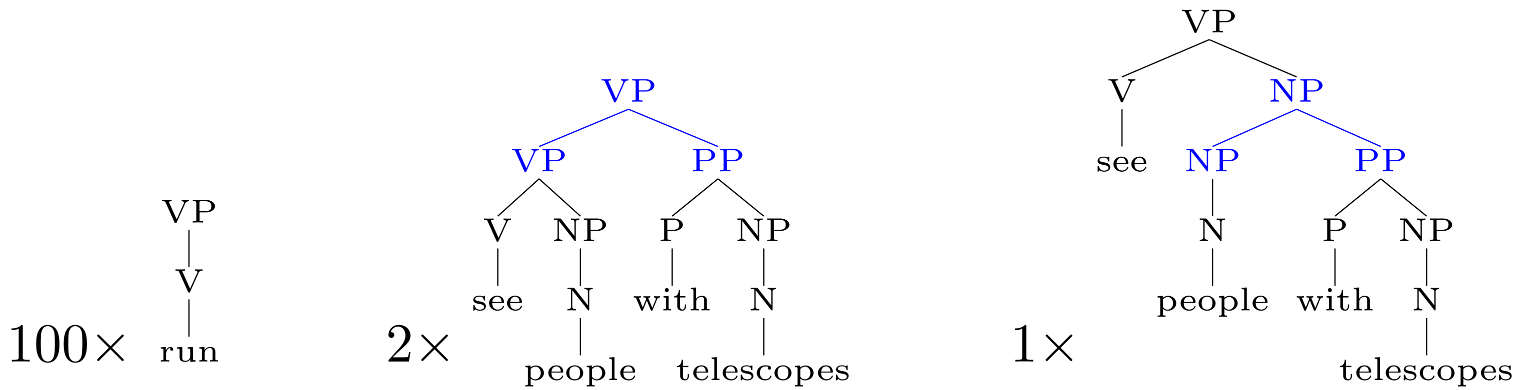
$$
V_w(t) \;=\; \exp \sum_j w_j f_j(t), \qquad Z_w(t) \;=\; \sum_{s' \in \mathcal{T}(s)} V_w(t')
$$

Conditional ML estimation

s	$f(t^\star)$	$\{f(t) : t \in \mathcal{T}(s), t \neq t^\star(s)\}$
sentence 1	$(1, 3, 2)$	$(2, 2, 3)$ $(3, 1, 5)$ $(2, 6, 3)$
sentence 2	$(7, 2, 1)$	$(2, 5, 5)$
sentence 3	$(2, 4, 2)$	$(1, 1, 7)$ $(7, 2, 1)$
...	...	...

- Parser designer specifies *feature functions* $f = (f_1, \ldots, f_m)$
- A parser produces trees $\mathcal{T}(s)$ for each sentence s
- Treebank tells us correct tree $t^\star(s) \in \mathcal{T}(s)$ for sentence s
- Feature functions f apply to each tree $t \in \mathcal{T}(s)$, producing feature values $f(t) = (f_1(t), \ldots, f_m(t))$
- MCLE estimates feature weights $w = (w_1, \ldots, w_m)$

Conditional vs joint MLE



$\ldots \times 2/105 \times \ldots$ $\qquad$ $\ldots \times 1/7 \times \ldots$

$\ldots \times 2/7 \times \ldots$ $\qquad$ $\ldots \times 1/7 \times \ldots$

Rule	**count**	**rel freq**	**rel freq**
VP $\rightarrow$ V	100	100/105	4/7
VP $\rightarrow$ V NP	3	3/105	1/7
VP $\rightarrow$ VP PP	2	2/105	**2/7**
NP $\rightarrow$ N	6	6/7	6/7
NP $\rightarrow$ NP PP	1	1/7	**1/7**

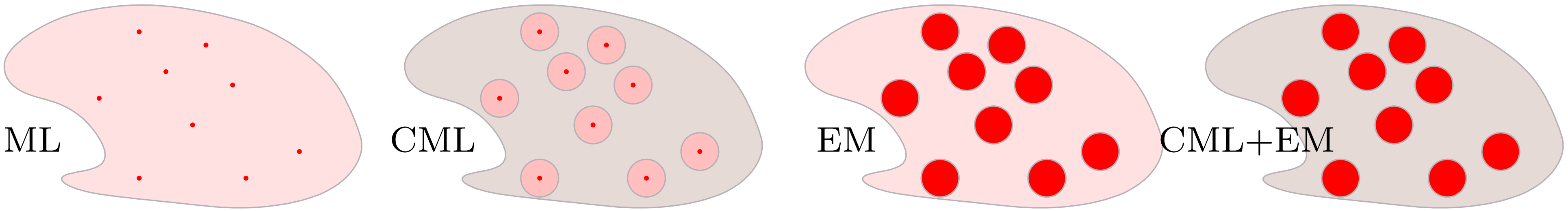
Conditional estimation

- The pseudo-partition function $Z_w(s)$ is *much easier to compute* than the partition function Z_w
  - Z_w requires a sum over $\mathcal{T}$
  - $Z_w(s)$ requires a sum over $\mathcal{T}(s)$ (parses of s)
- *Maximum likelihood* estimates full joint distribution
  - learns $\mathrm{P}(s)$ and $\mathrm{P}(t|s)$
- *Conditional ML* estimates a conditional distribution
  - learns $\mathrm{P}(t|s)$ but not $\mathrm{P}(s)$
  - conditional distribution is what you need for parsing
  - cognitively more plausible?
- Conditional estimation requires labelled training data: no obvious EM extension

CML estimation and hidden data

- Conditional ML estimation *ignores* distribution of strings

$\Rightarrow$ Cannot learn from strings alone



	maximizes likelihood of	**relative to**
MLE	t_i	$\mathcal{T}$
CMLE	t_i	$\mathcal{T}(s_i)$
EM	$\mathcal{T}(s_i)$	$\mathcal{T}$
CMLE+EM	$\mathcal{T}(s_i)$	$\mathcal{T}(s_i)$

Conditional estimation

	Correct parse's features	**All other parses' features**
sentence 1	$[1, 3, 2]$	$[2, 2, 3]$ $[3, 1, 5]$ $[2, 6, 3]$
sentence 2	$[7, 2, 1]$	$[2, 5, 5]$
sentence 3	$[2, 4, 2]$	$[1, 1, 7]$ $[7, 2, 1]$
$\ldots$	$\ldots$	$\ldots$

- Training data is *fully observed* (i.e., parsed data)
- Choose w to maximize (log) likelihood of *correct* parses relative to other parses
- Distribution of *sentences* is ignored
- *Nothing is learnt from unambiguous examples*
- Other discriminative learners solve this problem in different ways

Pseudo-constant features are uninformative

	Correct parse's features	**All other parses' features**
sentence 1	$[1, 3, 2]$	$[2, 2, 2]$ $[3, 1, 2]$ $[2, 6, 2]$
sentence 2	$[7, 2, 5]$	$[2, 5, 5]$
sentence 3	$[2, 4, 4]$	$[1, 1, 4]$ $[7, 2, 4]$
...	...	...

- *Pseudo-constant features* are identical within every set of parses
- They contribute the same constant factor to each parses' likelihood
- They do not distinguish parses of any sentence $\Rightarrow$irrelevant

Pseudo-maximal features ⇒ unbounded weights

	Correct parse's features	**All other parses' features**
sentence 1	$[1, 3, 2]$	$[2, 3, 4]$ $[3, 1, 1]$ $[2, 1, 1]$
sentence 2	$[2, 7, 4]$	$[3, 7, 2]$
sentence 3	$[2, 4, 4]$	$[1, 1, 1]$ $[1, 2, 4]$

- A *pseudo-maximal feature* always reaches its maximum value within a parse on the correct parse
- If f_j is pseudo-maximal, $\widehat{w_j}' \to \infty$ (hard constraint)
- If f_j is pseudo-minimal, $\widehat{w_j}' \to -\infty$ (hard constraint)

Regularization

- f_j is pseudo-maximal over training data $\not\Rightarrow$ f_j is pseudo-maximal over all strings (sparse data)
- With many more features than data, log-linear models can over-fit
- Regularization: add *bias* term to ensure $\widehat{w}'$ is finite and small
- In these experiments, the regularizer is a polynomial penalty term

$$\begin{aligned} \widehat{w}' &= \arg\max_{w} \log L'_D(w) - c \sum_{j=1}^{m} |w_j|^p \\ p &= 2 \text{ is } \textit{Gaussian prior}, p = 1 \text{ gives sparse solns} \end{aligned}$$

- $p = 2$ corresponds to Bayesian estimation with Gaussian prior $e^{-c\sum_j w_j^2}$

$$\begin{aligned} \mathrm{P}(M|D) &\propto \underbrace{\mathrm{P}(D|M)}_{\text{likelihood}} \underbrace{\mathrm{P}(M)}_{\text{prior}} \\ \log \mathrm{P}(M|D) &= \log \mathrm{P}(D|M) + \log \mathrm{P}(M) + a \end{aligned}$$

More on regularization

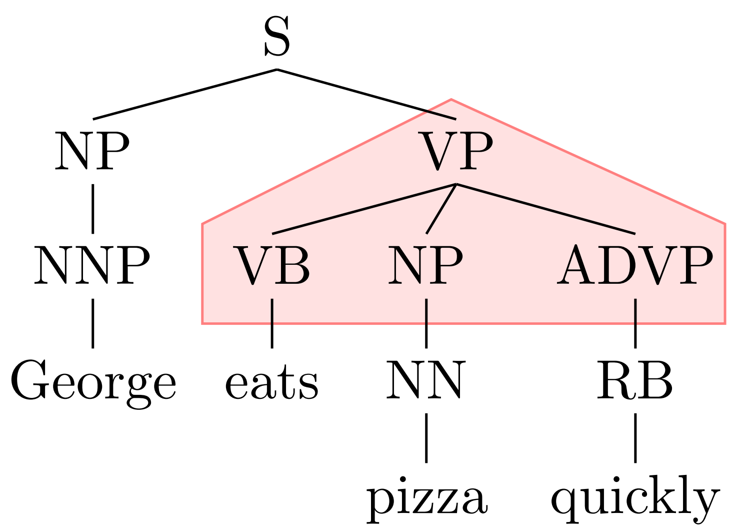
$$
\begin{aligned}
D &= ((s_1, t_1), \ldots, (s_n, t_n)), \text{ string } s_i, \text{ tree } t_i \\
Q(w) &= \sum_{i=1}^{n} \log \mathrm{P}(t_i|w) - c \log \sum_{j=1}^{m} |w_j|^p \\
\frac{\partial Q}{\partial w_j} &= \underbrace{\sum_{i=1}^{n} f_j(t_i) - \sum_{i=1}^{n} \mathrm{E}[f_j|s_i]}_{\text{likelihood}} - \underbrace{cp|x_j|^{p-1}}_{\text{prior}} \\
\frac{\partial Q}{\partial w_j} = 0 &\Rightarrow \sum_{i=1}^{n} f_j(t_i) = \sum_{i=1}^{n} \mathrm{E}[f_j|s_i] + cp|x_j|^{p-1}
\end{aligned}
$$

Optimization algorithms for finding CMLE

- Specialized algorithms: Iterative scaling and various enhancements
- General purpose numerical algorithms that use gradient: Conjugate gradient, Limited Memory Variable Metric
  - numerical analysts have spent years optimizing algorithms
  - good general purpose optimization packages are freely downloadable
- Most time is spent calculating likelihood of each tree in training data
  - since you're visiting each tree, might as well calculate derivative as well
- Currently LMVM is fastest method for parsing problems

Comparing MLE and CMLE in PCFG parsing

- MLE is relative frequency estimator (involves counting rule occurences in training trees)



$$\widehat{\mathrm{P}}(\mathrm{VP} \rightarrow \ \mathrm{VB\ NP\ ADVP}) \quad = \quad \frac{C(\mathrm{VP} \rightarrow \ \mathrm{VB\ NP\ ADVP})}{\sum_{\alpha \mathrm{s.t.VP} \rightarrow \alpha} C(\mathrm{VP} \rightarrow \ \alpha)}$$

Comparing estimators: PCFG parsing

- MCLE involves maximizing a complex non-linear function
  - $\partial Z_w(s)/\partial w_j$ involves $\mathrm{E}_w[f_j|s]$ (expected number of times rule j appears in training data)
    - computed using *inside-outside algorithm*
  - conjugate gradient (iterative optimization)
  - each iteration involves summing over all parses of each training sentence

⇒ Use the small ATIS treebank corpus
  - Trained on 1088 sentences of ATIS1 corpus
  - Tested on 294 sentences of ATIS2 corpus

- MCLE estimator initialized with MLE probabilities

PCFG parsing results

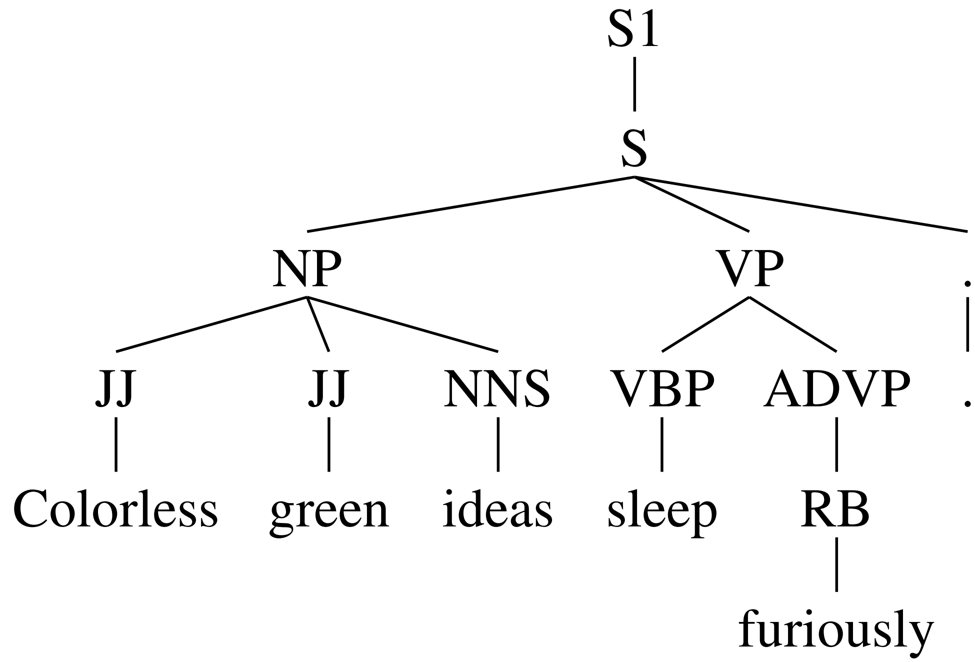
	MLE	MCLE
$-\log$ likelihood of training data	13857	13896
$-\log$ *conditional* likelihood of training data	1833	1769
$-\log$ *marginal* probability of training strings	12025	12127
Labelled precision of test data	0.815	0.817
Labelled recall of test data	0.789	0.794

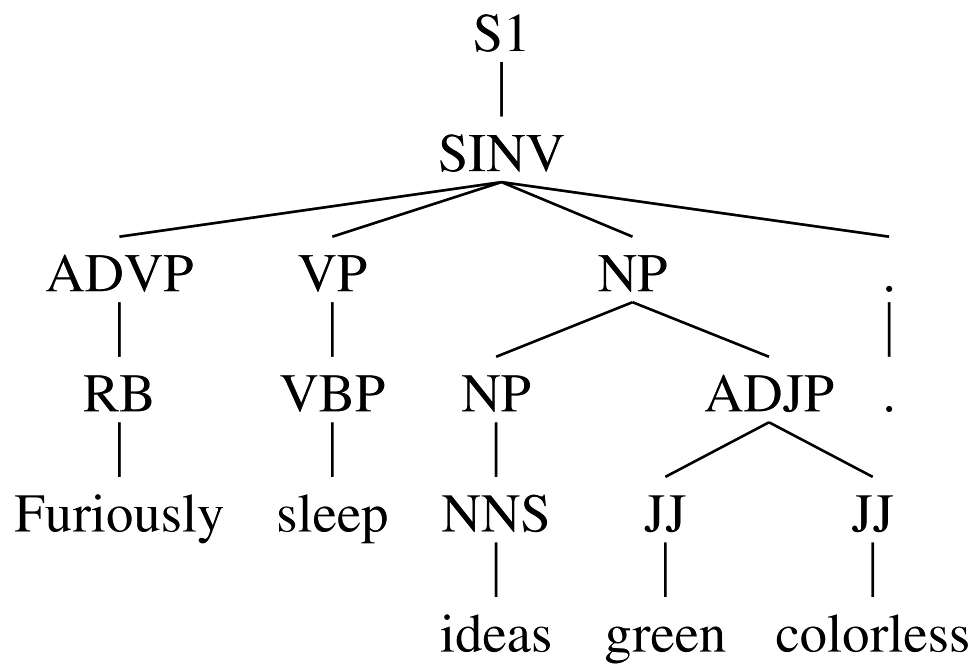
- Precision/recall difference *not significant* ($p \approx 0.1$)

SWITCH TO CoNLL 2005 talk here

Conclusion

- It's possible to build (moderately) accurate, broad-coverage parsers
- Generative parsing models are easy to estimate, but make questionable independence assumptions
- Exponential models don't assume independence, so it's easy to add new features, but are difficult to estimate
- Coarse-to-fine conditional MLE for exponential models is a compromise
  - flexibility of exponential models
  - possible to estimate from treebank data
- Gives the currently best-reported parsing accuracy results


S1
S
NP
VP
.
JJ
JJ
NNS
VBP
ADVP
.
Colorless
green
ideas
sleep
RB
furiously


S1
SINV
ADVP
VP
NP
.
RB
VBP
NP
ADJP
.
Furiously
sleep
NNS
JJ
JJ
ideas
green
colorless