Grammars, graphs and automata

Mark Johnson

Brown University

ALTA summer school

December 2003

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

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

Motivation

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

- we hope that the essence of these processes (in humans and machines) is the computational manipulation of information

Natural language processing is the use of computers for processing natural language text and speech

- Machine translation
- Information extraction
- Question-answering

Why grammars?

- A *grammar* describes a language
  - usually specifies its sentences and provides descriptions of them (e.g., their meanings)
- *Parsing* is the process of identifying the sentence's description
- *Generation* is the process of translating meanings into *grammatical* or well-formed sentences
- *Phrase-structure grammars* describe how words group into phrases
  - provides a tree or graph representation of each sentence's structure
- *Grammars provide a general-purpose computational framework*
  - More general than most *finite state automata*
  - Complementary with *graphical models* (esp. plates)

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

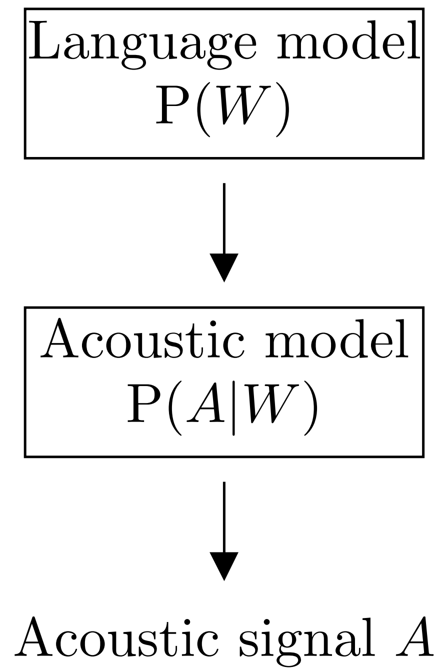
- A *probability distribution* over a countable set Ω is a function $\mathrm{P} : \Omega \to [0,1]$ which satisfies $1 = \sum_{\omega \in \Omega} \mathrm{P}(\omega)$.
- A *random variable* is a function $X : \Omega \to \mathcal{X}$. $\mathrm{P}(X{=}x) = \sum_{\omega : X(\omega)=x} \mathrm{P}(\omega)$
- 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

$$\widehat{w}(a) = \operatorname*{argmax}_{w} \mathrm{P}(W = w | A = a)$$

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



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

$$\begin{aligned}\widehat{w}(a) &= \operatorname*{argmax}_{w} \frac{\mathrm{P}(W = w)\mathrm{P}(A = a|W = w)}{\mathrm{P}(A = a)} \\ &= \operatorname*{argmax}_{w} \mathrm{P}(W = w)\mathrm{P}(A = a|W = w)\end{aligned}$$

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* $\mathrm{P}(W)$

Bayes nets

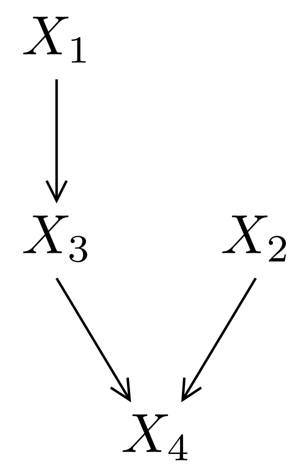
A Bayes net is a directed acyclic graph that depicts a way of factorizing a joint probability distribution into a product of conditional distributions.

Example: By Bayes rule:

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

But if $\mathrm{P}(X_i|X_1, \ldots, X_{i-1})$ doesn't depend on all of $X_1, \ldots, X_i$, then we can simplify this to something like:

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



Bayes nets depict such simplified products of conditionals.

- The Bayes net has a node for each variable.
- If the product contains a term $\mathrm{P}(X_i| \ldots, X_j, \ldots)$ then the Bayes net has an arc from j to i.

Marginalizing over a variable

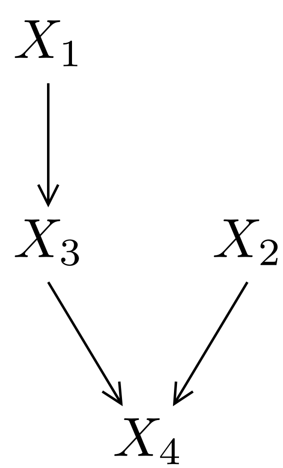
Marginalizing over a variable (i.e., summing over all of its possible values) deletes the node and connects all of its ancestors with all of its descendants

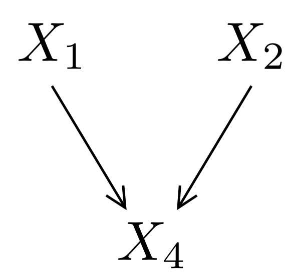
Example:

$$P(X_1, X_2, X_3, X_4) = P(X_1)P(X_2)P(X_3|X_1)P(X_4|X_2, X_3)$$

Marginalize over X_3, i.e.,

$$\begin{aligned} P(X_1, X_2, X_4) &= P(X_1)P(X_2)\sum_{X_3} P(X_3|X_1)P(X_4|X_2, X_3) \\ &= P(X_1)P(X_2)P(X_4|X_1, X_2) \end{aligned}$$





Conditioning on a variable

Conditioning on a variable (i.e., fixing its value) deletes the node and all links to it.

Example:

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

Set $X_3 = c$. Then

$$\mathrm{P}(X_1, X_2, X_4|X_3 = c) \;\propto\; \mathrm{P}(X_1)\mathrm{P}(X_2)\mathrm{P}(c|X_1)\mathrm{P}(X_4|X_2, c)$$

$$\begin{tikzcd}
X_1 \arrow[d] & & \\
X_3 \arrow[dr] & & X_2 \arrow[dl] \\
& X_4 &
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
X_1 & & X_2 \arrow[dl] \\
& X_4 &
\end{tikzcd}$$

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 $w_1 \ldots w_n$ based on the word pairs (w_i, w_{i+1}) the string contains.

A bigram model is a homogenous Markov chain:

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

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

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 $p_m(\$|\$) = 1$

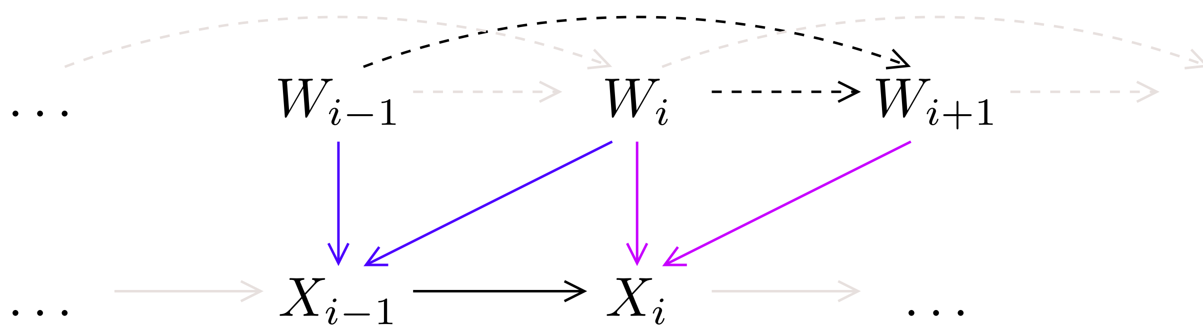
$\mathrm{P}(\text{Howard hates brocolli \$})$
$= \; p_s(\text{Howard}) p_m(\text{hates}|\text{Howard}) p_m(\text{brocolli}|\text{hates}) p_m(\$|\text{brocolli})$

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:

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



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

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.

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

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

Applications of homogeneous HMMs

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

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

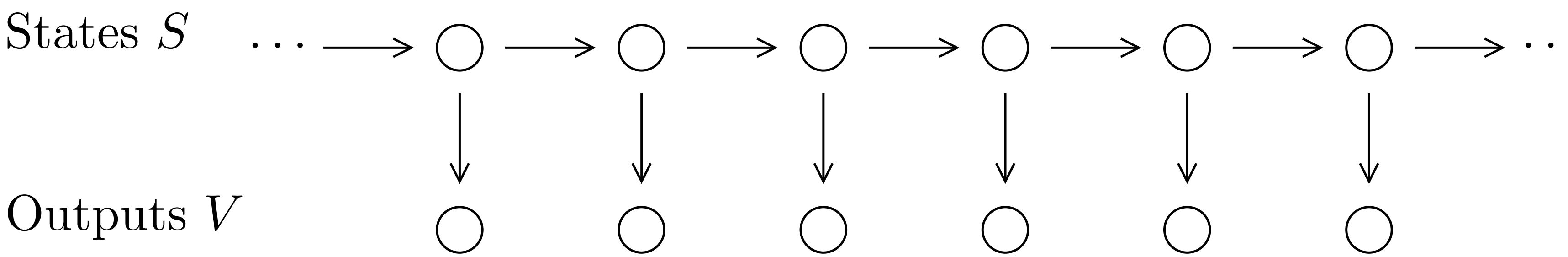
$$\begin{tikzcd}
\ldots \arrow[r] & S_{i-1} \arrow[r] \arrow[d] & S_i \arrow[r] \arrow[d] & S_{i+1} \arrow[r] \arrow[d] & \ldots \\
 & V_{i-1} & V_i & V_{i+1} &
\end{tikzcd}$$

Part of speech tagging:

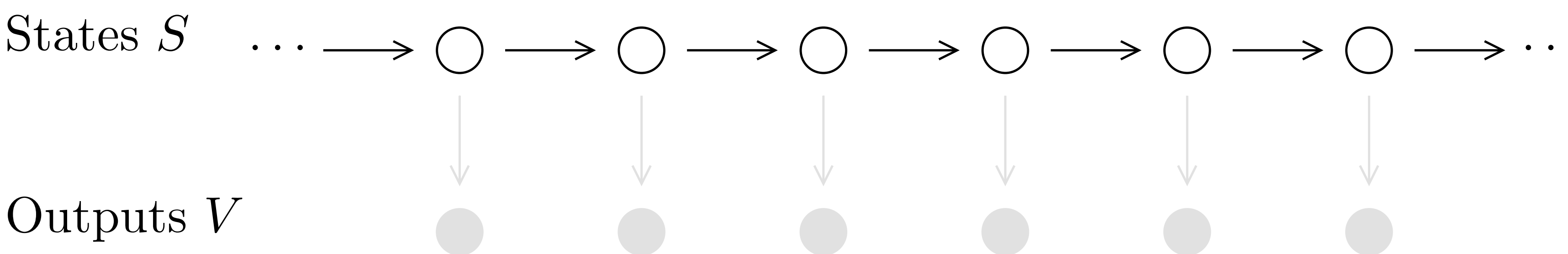
States are *parts of speech*, outputs are *words*

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

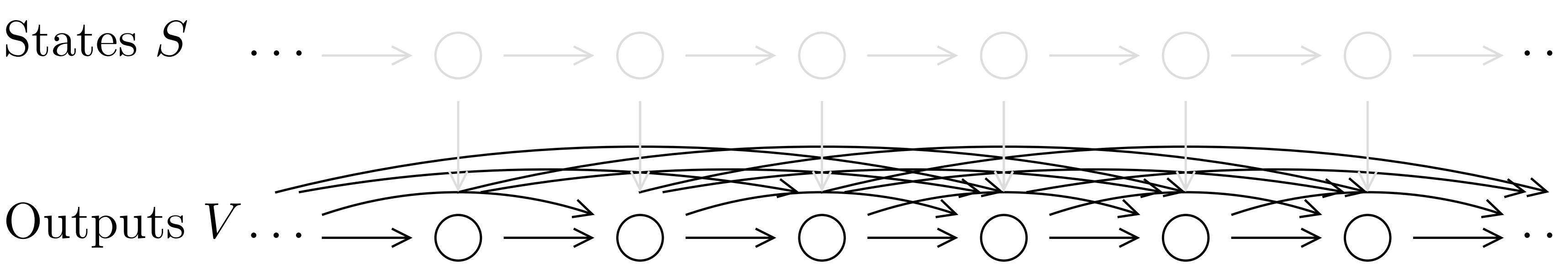
Properties of HMMs



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*



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.,

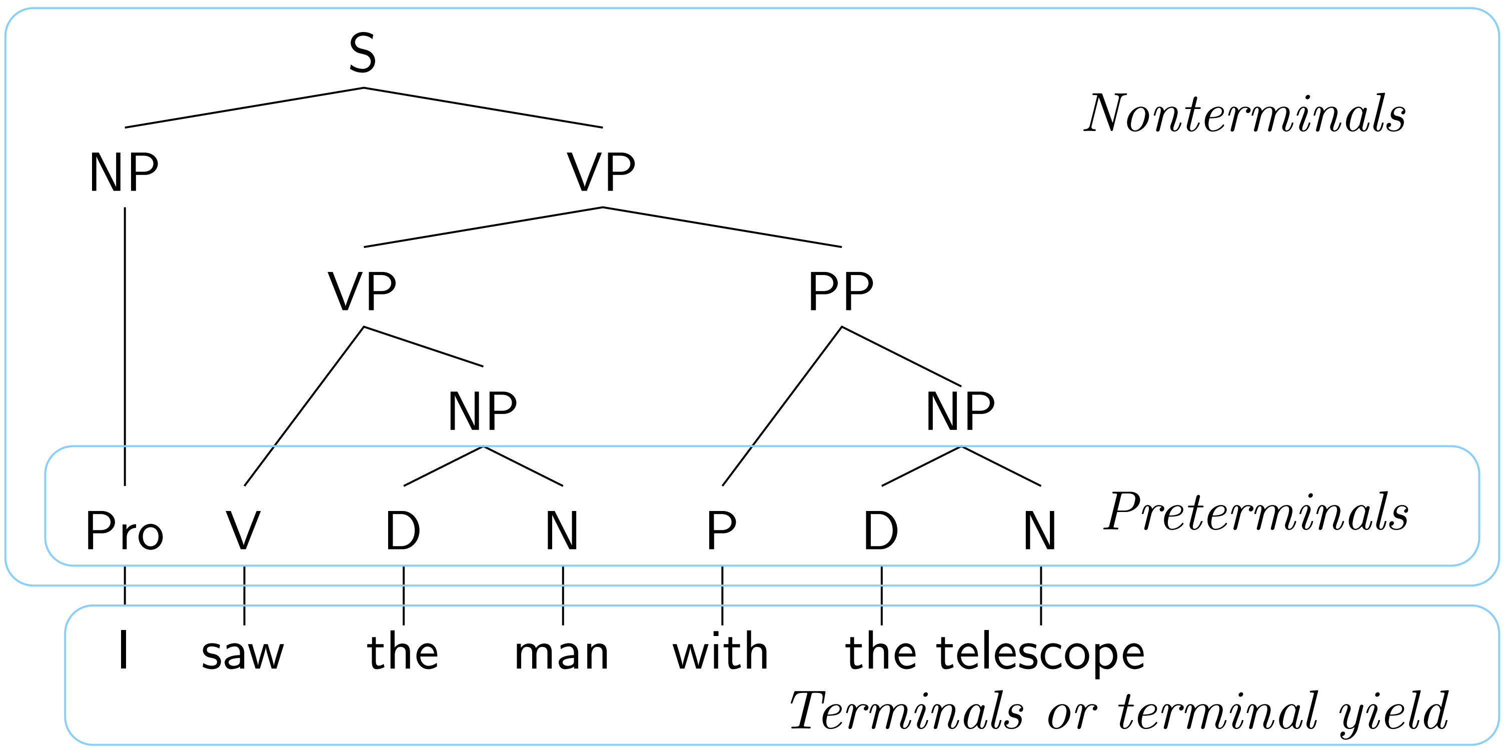
- $\forall w \in \mathcal{V}^\star \; 0 \leq \mathrm{P}(w) \leq 1$
- $\sum_{w \in \mathcal{V}^\star} \mathrm{P}(w) = 1$, i.e., P is *normalized*

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

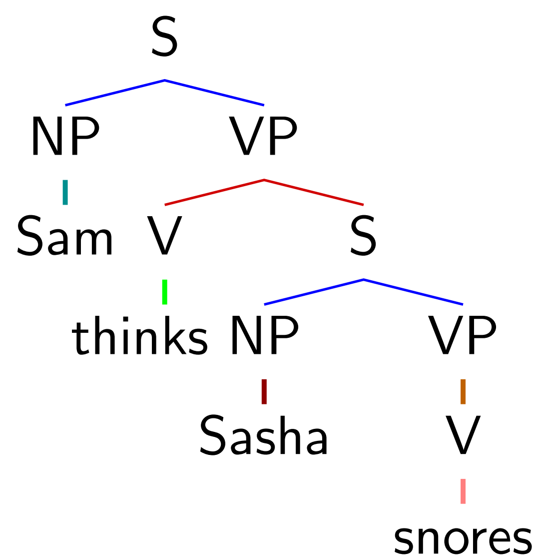
A *context-free grammar* $G = (\mathcal{V}, \mathcal{S}, s, \mathcal{R})$ consists of:

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

G *generates* a tree ψ iff

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

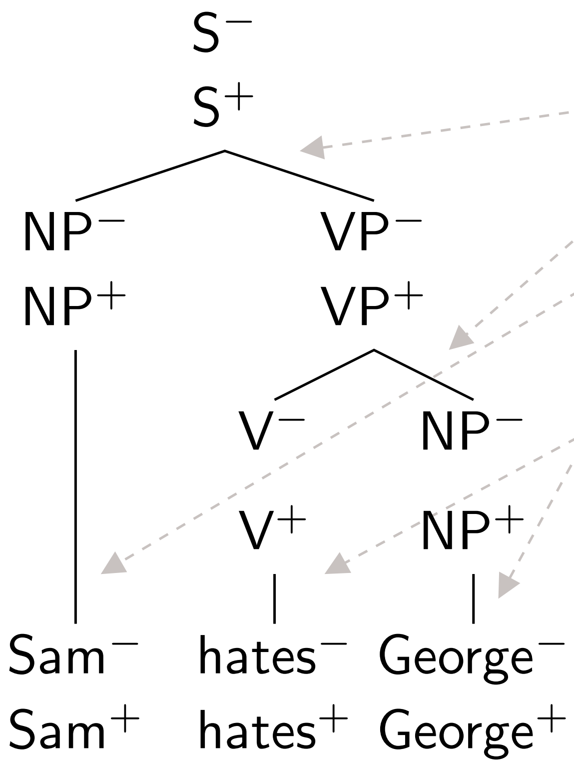
G generates a string $w \in \mathcal{V}^\star$ iff w is the *terminal yield* of a tree generated by G



Productions

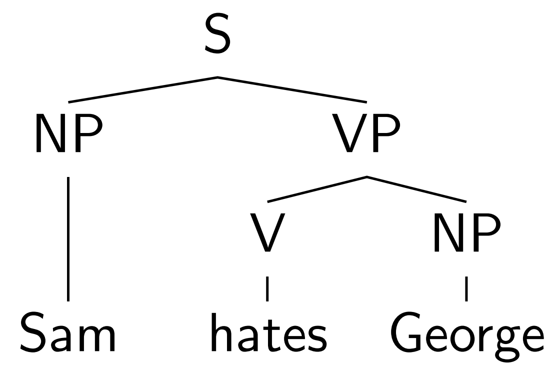
S → NP VP
NP → Sam
NP → Sasha
VP → V
VP → V S
V → thinks
V → snores

CFGs as "plugging" systems



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

Productions



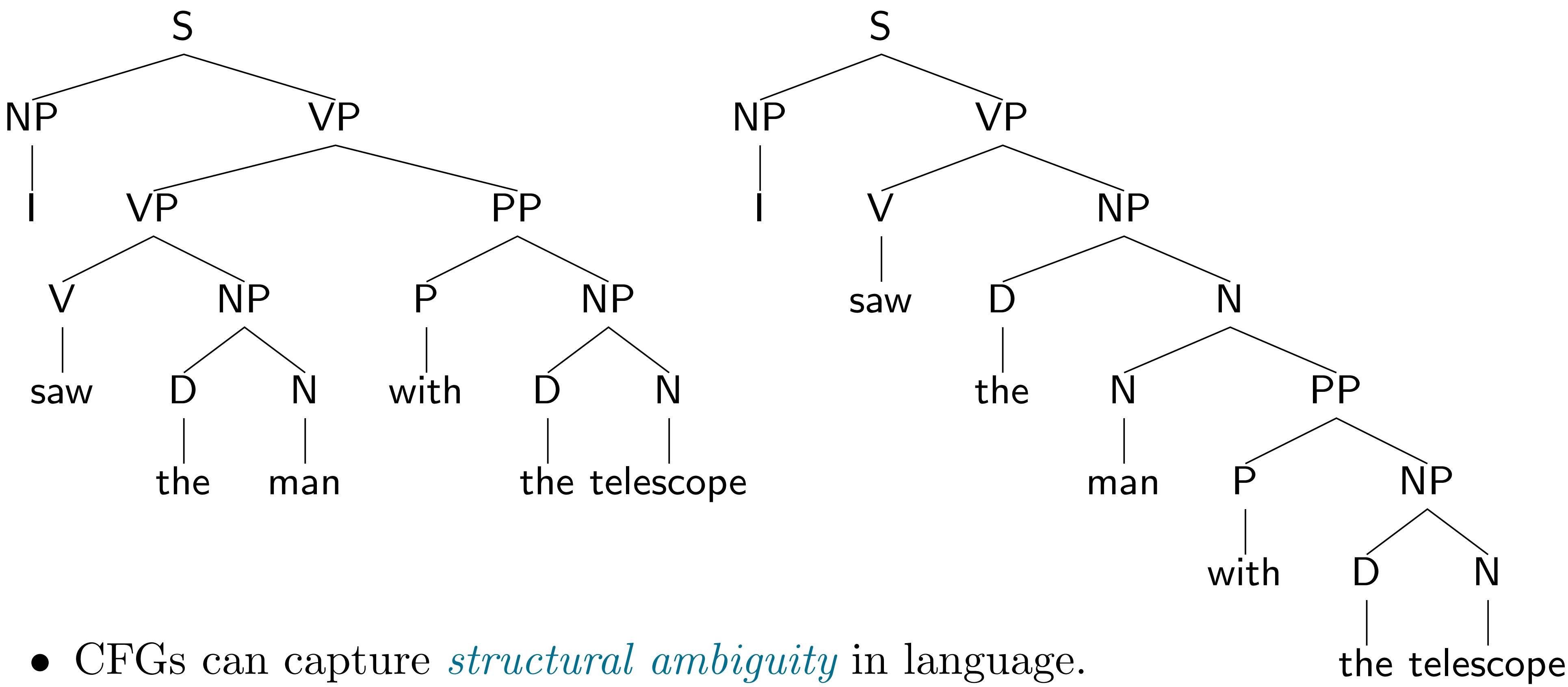
"Pluggings"

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

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



- 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 $A \to \beta \in \mathcal{R}$ 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 $s \Rightarrow_G \ldots \Rightarrow_G w$. $\Rightarrow_G^\star$ is the *reflexive and transitive closure* of $\Rightarrow_G$.

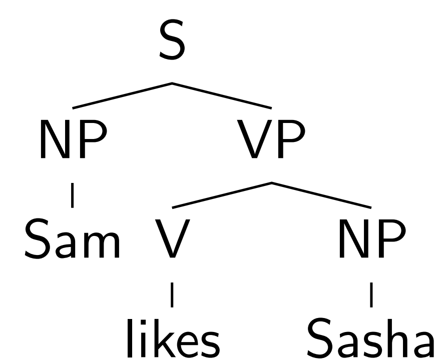
The *language generated* by G is $\{w : s \Rightarrow^\star w, w \in \mathcal{V}^\star\}$.

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

$$
\begin{aligned}
&\mathsf{S}\\
\Rightarrow\ &\mathsf{NP\ VP}\\
\Rightarrow\ &\mathsf{NP\ V\ NP}\\
\Rightarrow\ &\mathsf{Sam\ V\ NP}\\
\Rightarrow\ &\mathsf{Sam\ V\ Sasha}\\
\Rightarrow\ &\mathsf{Sam\ likes\ Sasha}
\end{aligned}
$$

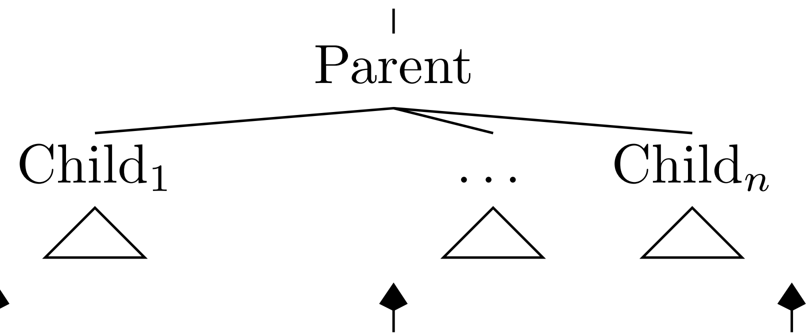
Steps in a *terminating derivation* are always *cuts in a parse tree*

Left-most and *right-most* derivations are unique



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 (1)

Leftmost derivation

S

S

Productions

S $\longrightarrow$ NP VP

NP $\longrightarrow$ D N

D $\longrightarrow$ no

N $\longrightarrow$ politican

VP $\longrightarrow$ V

V $\longrightarrow$ lies

Top-down parses are *left-most* derivations (2)

S

NP VP

Leftmost derivation

S

NP VP

Productions

S → NP VP

NP → D N

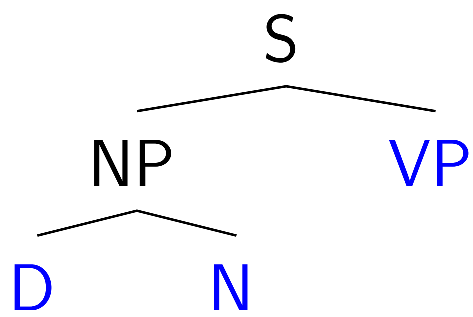
D → no

N → politican

VP → V

V → lies

Top-down parses are *left-most* derivations (3)



Leftmost derivation

S
NP VP
D N VP

Productions

S → NP VP

NP → D N

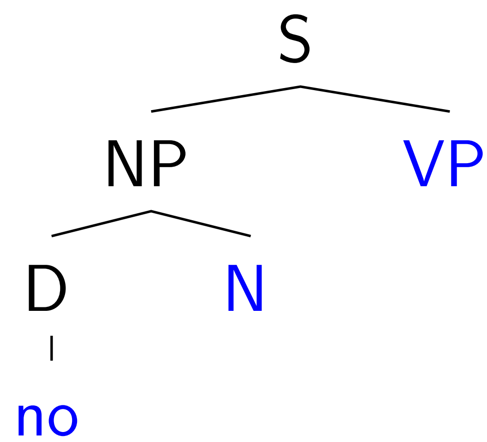
D → no

N → politican

VP → V

V → lies

Top-down parses are *left-most* derivations (4)



Leftmost derivation

S
NP VP
D N VP
no N VP

Productions

S → NP VP

NP → D N

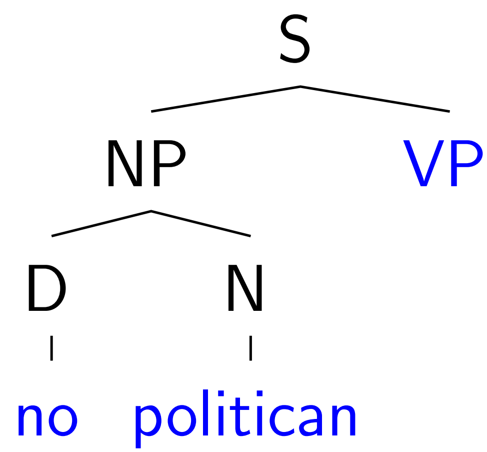
D → no

N → politican

VP → V

V → lies

Top-down parses are *left-most* derivations (5)



Productions

S $\longrightarrow$ NP VP

NP $\longrightarrow$ D N

D $\longrightarrow$ no

N $\longrightarrow$ politican

VP $\longrightarrow$ V

V $\longrightarrow$ lies

Leftmost derivation

S

NP VP

D N VP

no N VP

no politican VP

Top-down parses are *left-most* derivations (6)

Productions

S → NP VP

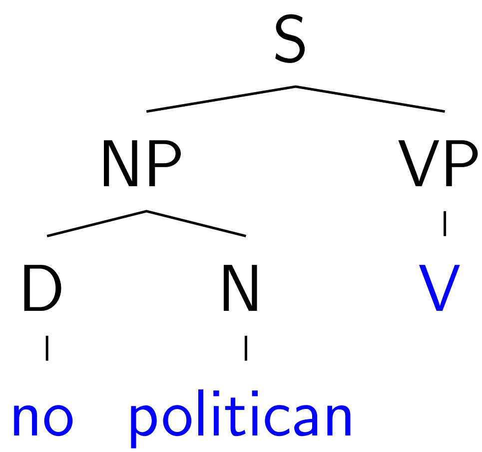
NP → D N

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 (7)

Productions

S $\longrightarrow$ NP VP

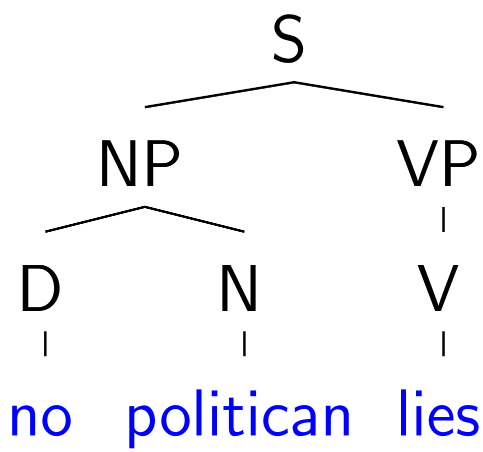
NP $\longrightarrow$ D N

D $\longrightarrow$ no

N $\longrightarrow$ politican

VP $\longrightarrow$ V

V $\longrightarrow$ 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 rightmost-most* derivations (1)

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 rightmost-most* derivations (2)

Rightmost derivation

Productions

S $\rightarrow$ NP VP

NP $\rightarrow$ D N

D $\rightarrow$ no

N $\rightarrow$ politican

VP $\rightarrow$ V

V $\rightarrow$ lies

D

no politican lies

D politican lies
no politican lies

Bottom-up parses are *reversed rightmost-most* derivations (3)

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 rightmost-most* derivations (4)

Rightmost derivation

Productions

S → NP VP

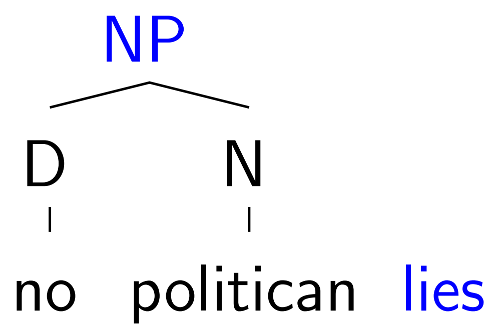
NP → D N

D → no

N → politican

VP → V

V → lies



NP lies
D N lies
D politican lies
no politican lies

Bottom-up parses are *reversed rightmost-most* derivations (5)

Rightmost derivation

Productions

S → NP VP

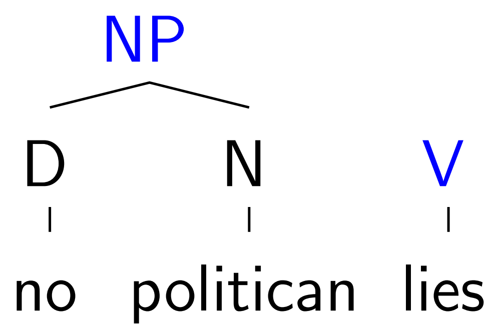
NP → D N

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 rightmost-most* derivations (6)

Productions

S → NP VP

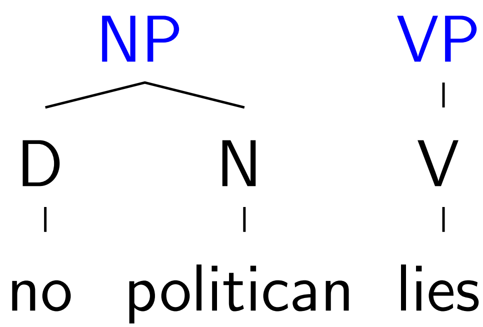
NP → D N

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 rightmost-most* derivations (7)

Productions

S → NP VP

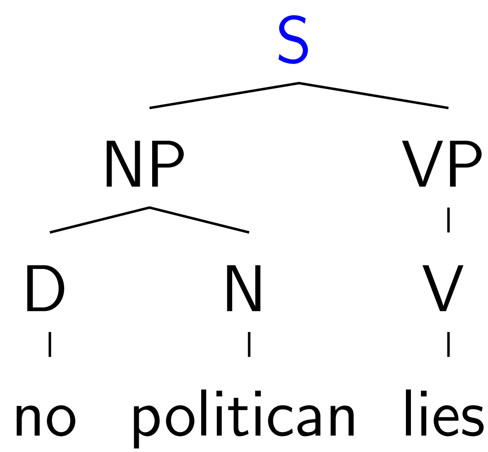
NP → D N

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

$$P\left(\begin{array}{c} \textsf{S} \\ \begin{array}{cc} \textsf{NP} & \textsf{VP} \\ | & | \\ \textsf{George} & \textsf{V} \\ & | \\ & \textsf{barks} \end{array} \end{array}\right) = 0.45 \qquad P\left(\begin{array}{c} \textsf{S} \\ \begin{array}{cc} \textsf{NP} & \textsf{VP} \\ | & | \\ \textsf{Al} & \textsf{V} \\ & | \\ & \textsf{snores} \end{array} \end{array}\right) = 0.1$$

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'

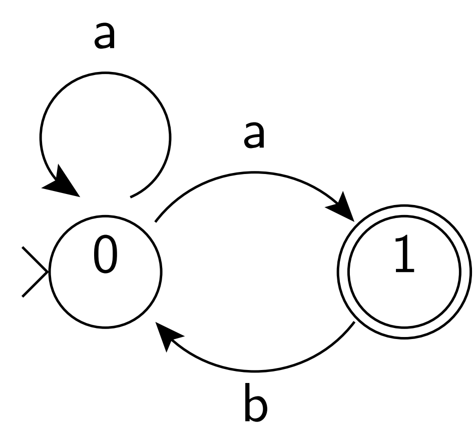
Moore automata (homogenous HMMs) choose x based on s' alone

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:

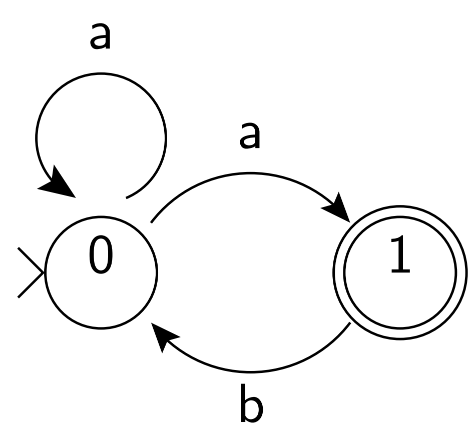
- terminals $\mathcal{V}$, states $\mathcal{S}$ and start state $s_0 \in \mathcal{S}$ as before,
- $p_f(s)$, the probability of *halting at state* $s \in \mathcal{S}$, and
- $p_m(v, s'|s)$, the probability of *moving from* $s \in \mathcal{S}$ *to* $s' \in \mathcal{S}$ *and emitting a* $v \in \mathcal{V}$.

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

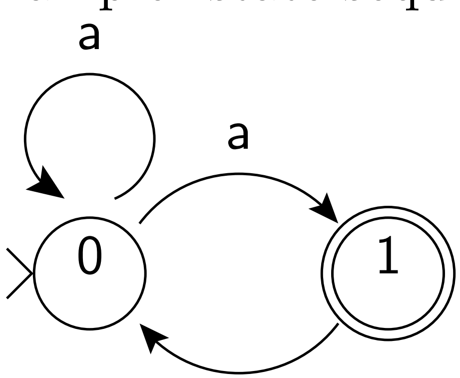


Bayes net representation of Mealy PFSA

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



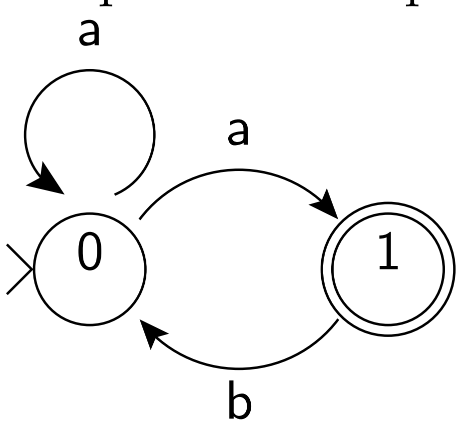
Mealy FSA

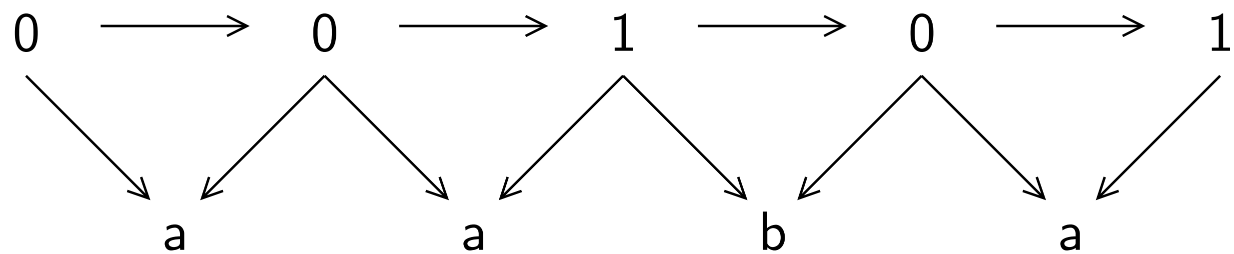
$$
\begin{tikzcd}
0 \arrow[rr] \arrow[dr] & & 0 \arrow[rr] \arrow[dl] \arrow[dr] & & 1 \arrow[rr] \arrow[dl] \arrow[dr] & & 0 \arrow[rr] \arrow[dl] \arrow[dr] & & 1 \arrow[dl] \\
& \mathsf{a} & & \mathsf{a} & & \mathsf{b} & & \mathsf{a} &
\end{tikzcd}
$$

Bayes net for aaba

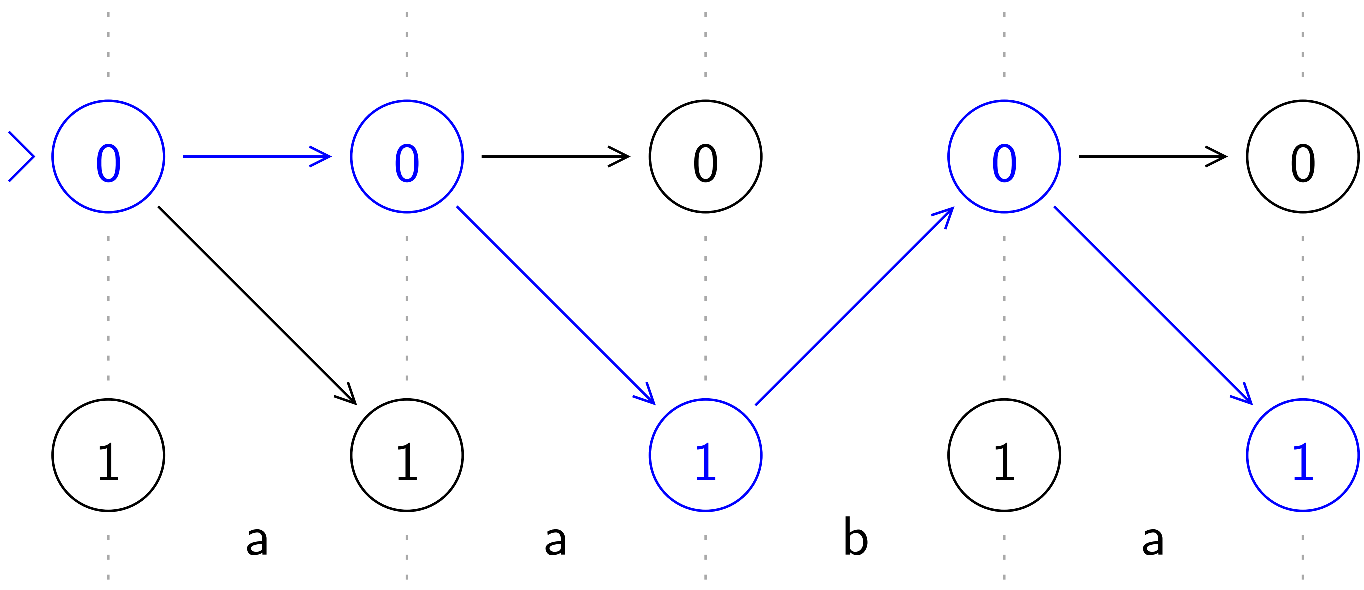
The trellis for a Mealy PFSA

Example: state sequence 00101 for string aaba





Bayes net for aaba

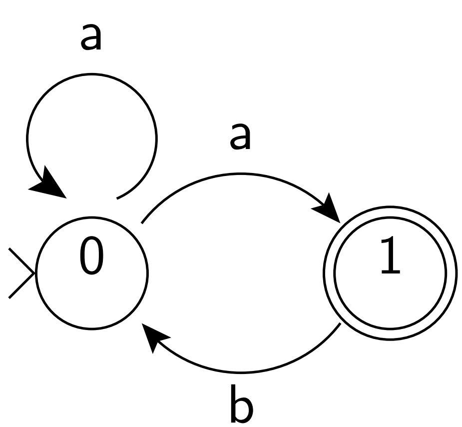


Probabilistic Mealy FSA as PCFGs

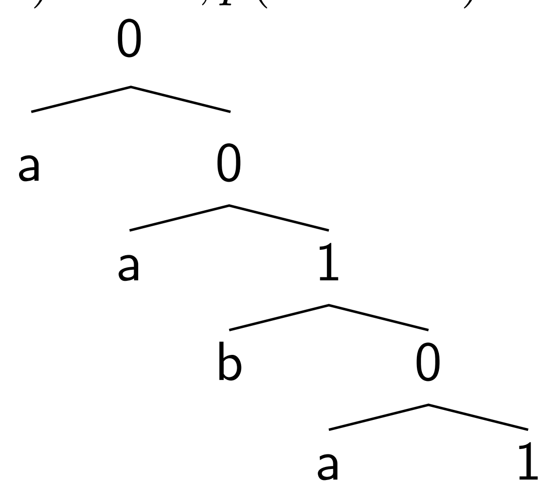
Given a Mealy PFSA $M = (\mathcal{V}, \mathcal{S}, s_0, p_f, p_m)$, let G_M have the same terminals, states and start state as M, and have productions

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

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



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.

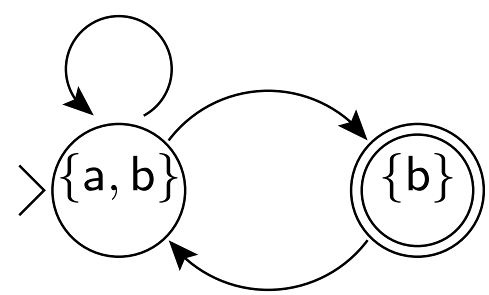
A *Moore finite state automaton* $M = (\mathcal{V}, \mathcal{S}, s_0, \mathcal{F}, \mathcal{M}, \mathcal{L})$ is composed of:

- $\mathcal{V}$, $\mathcal{S}$, 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

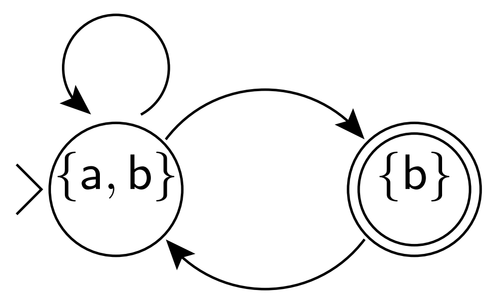
A *probabilistic Moore automaton* $M = (\mathcal{V}, \mathcal{S}, s_0, p_f, p_m, p_\ell)$ consists of:

- 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

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



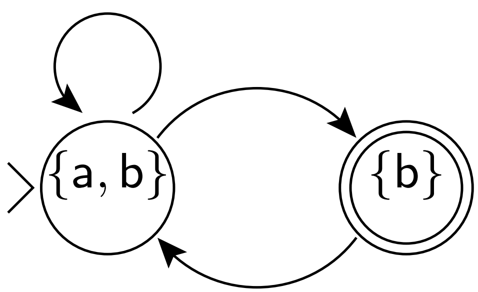
Example: $p_f(\mathsf{0}) = 0, p_f(\mathsf{1}) = 0.1,$
$p_\ell(\mathsf{a}|\mathsf{0}) = 0.4, p_\ell(\mathsf{b}|\mathsf{0}) = 0.6, p_\ell(\mathsf{b}|\mathsf{1}) = 1,$
$p_m(\mathsf{0}|\mathsf{0}) = 0.2, p_m(\mathsf{1}|\mathsf{0}) = 0.8, p_m(\mathsf{0}|\mathsf{1}) = 0.9$
$P_M(\mathsf{0101}, \mathsf{bab})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

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)

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

Example: state sequence 0101 for string bab



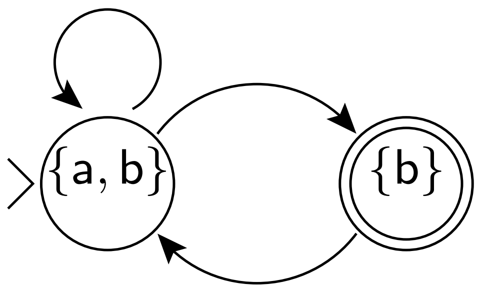
Moore FSA

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

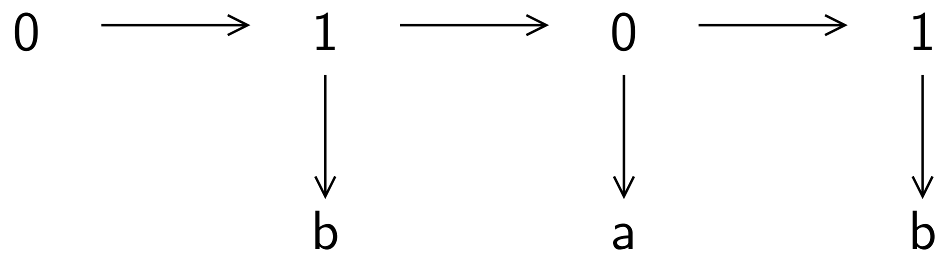
Bayes net for bab

Trellis representation of Moore PFSA

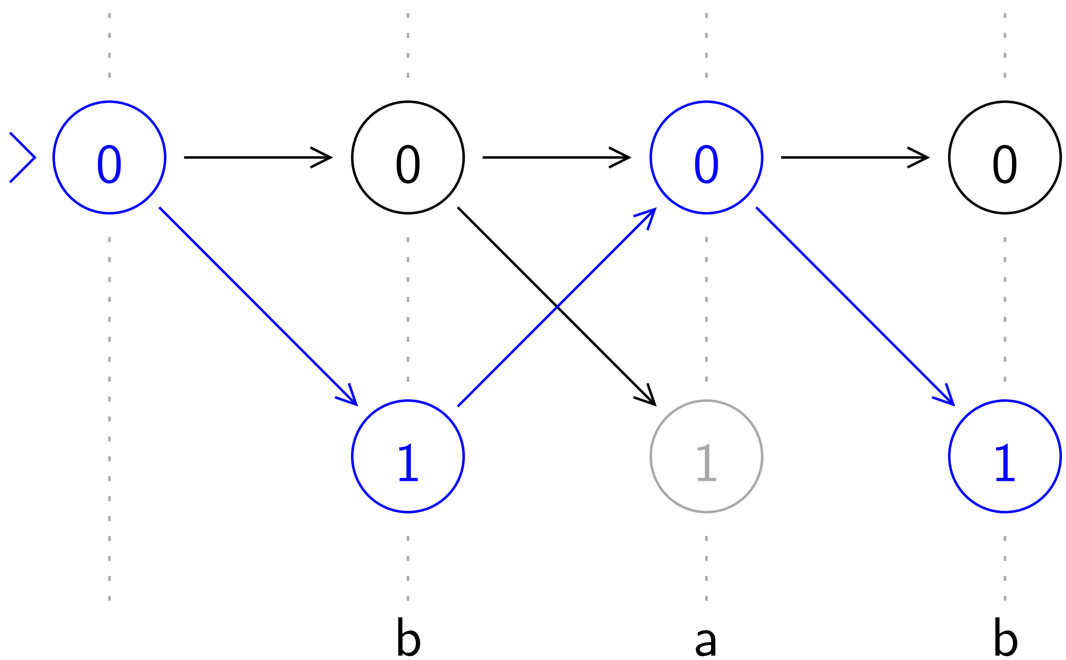
Example: state sequence 0101 for string bab



Moore FSA



Bayes net for bab

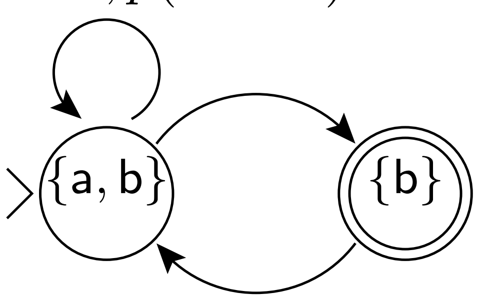


Probabilistic Moore FSA as PCFGs

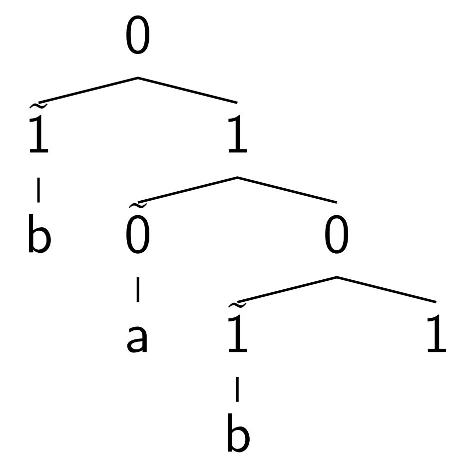
Given a Moore PFSA $M = (\mathcal{V}, \mathcal{S}, s_1, p_f, p_m, p_\ell)$, let G_M have the same terminals and start state as M, *two nonterminals s and $\tilde{s}$ for each state $s \in \mathcal{S}$*, and productions

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

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



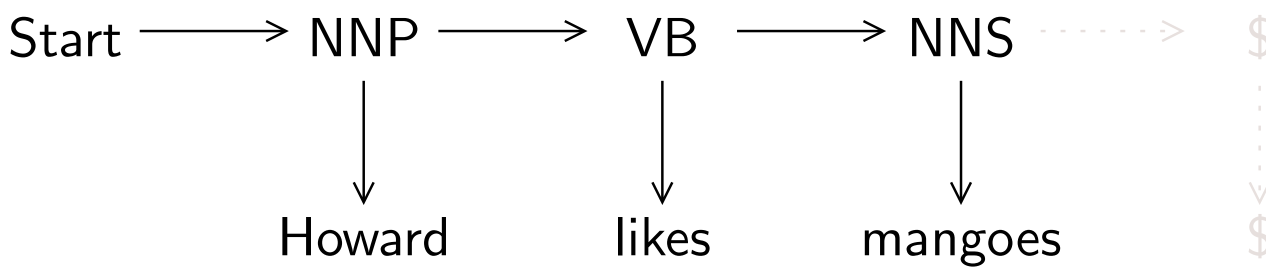
Moore FSA

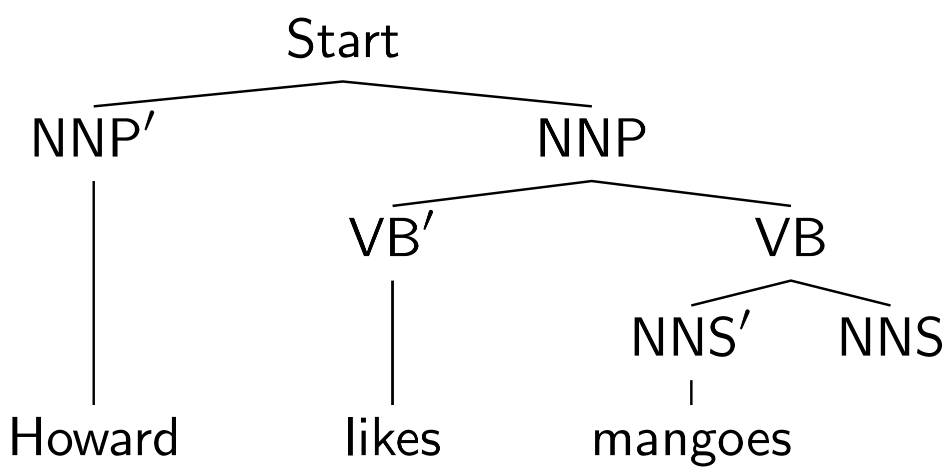


PCFG parse of bab

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 $|\mathcal{V}| \approx 2 \times 10^4$

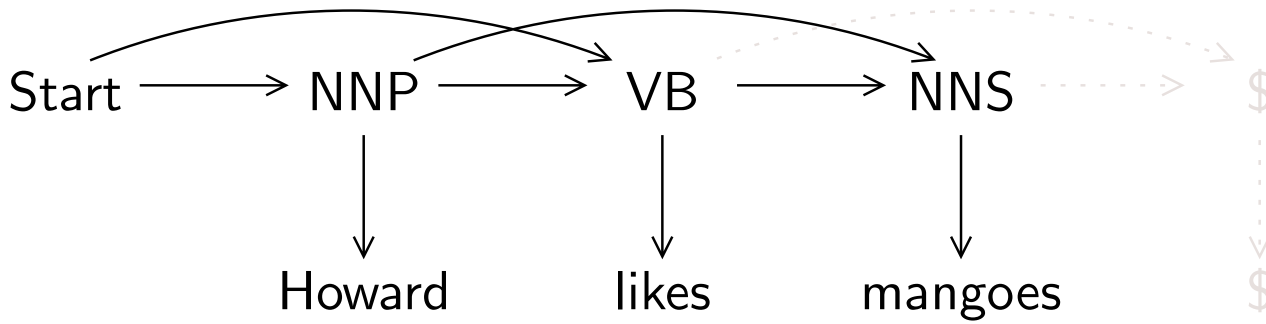
- A Mealy automaton has $\approx 5 \times 10^7$ parameters
- A Moore automaton has $\approx 10^6$ parameters

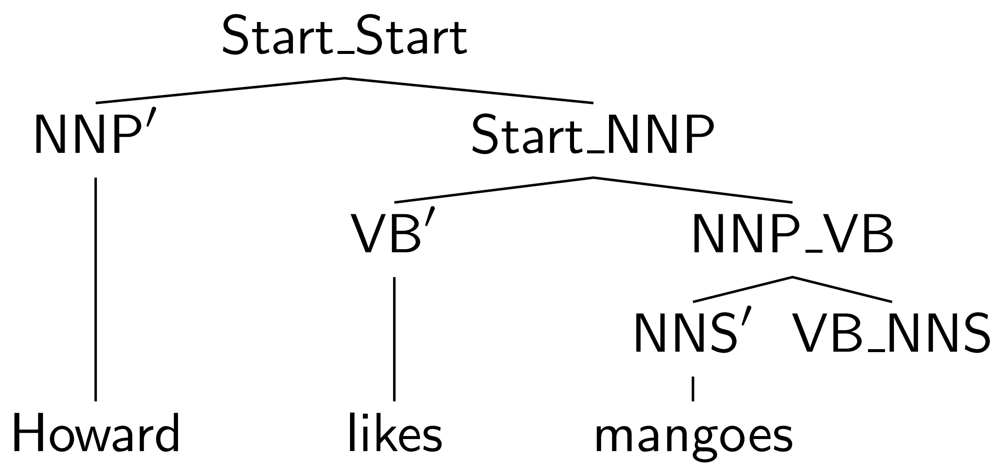
A Moore automaton seems more reasonable for POS-tagging

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 $v \in \mathcal{V}$

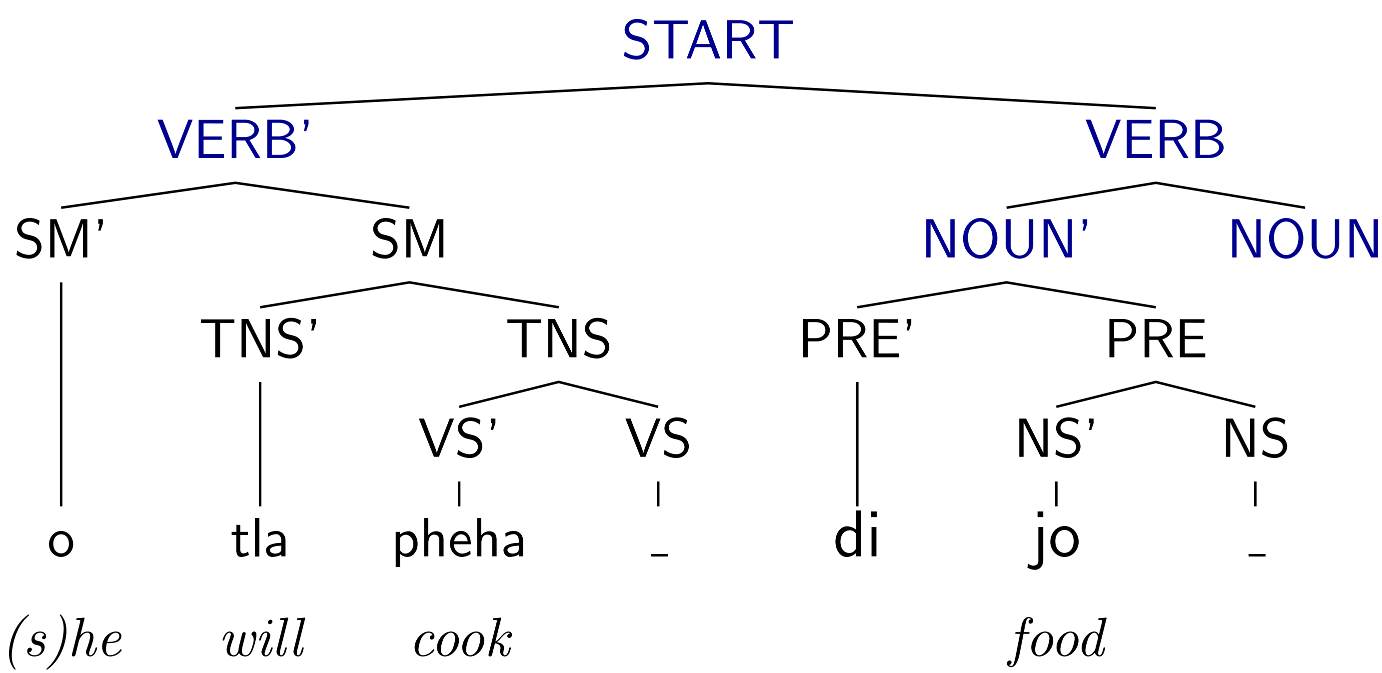
Advantages of using grammars

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 $A \to \beta$ or $A \to \beta\, B$ 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 $A \to \beta$ or $A \to B\, \beta$ 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*

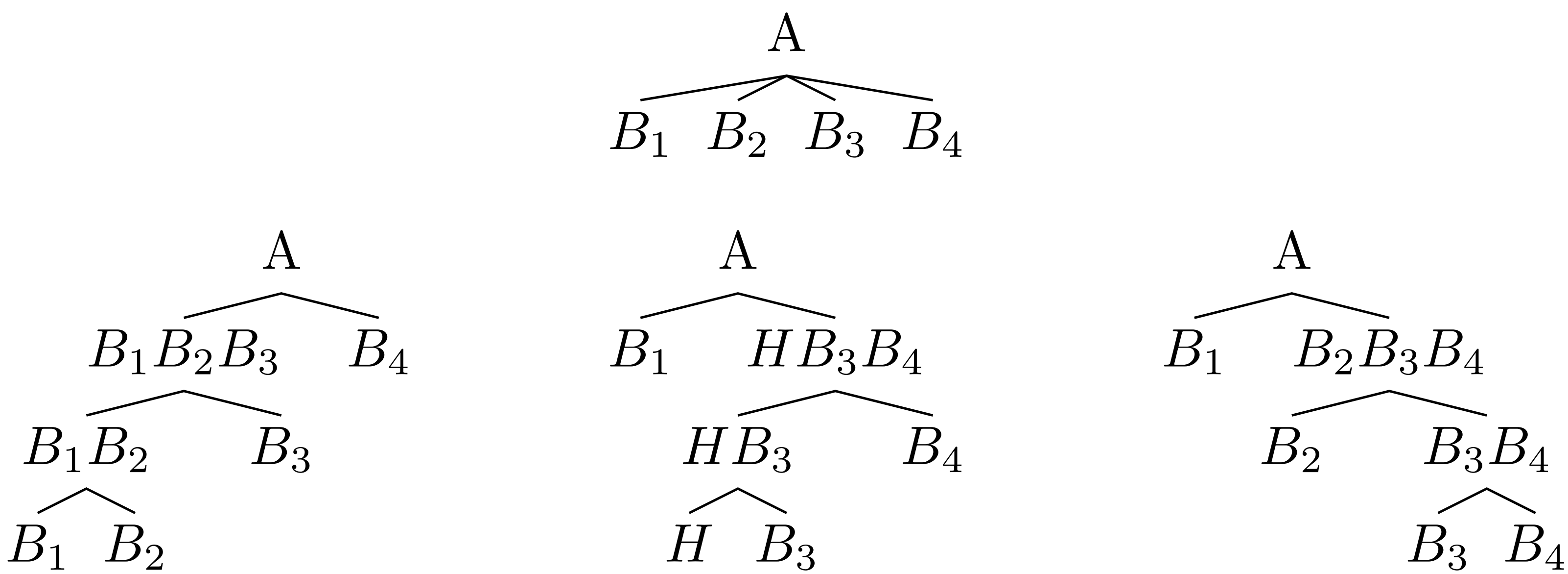
Topics

- 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.

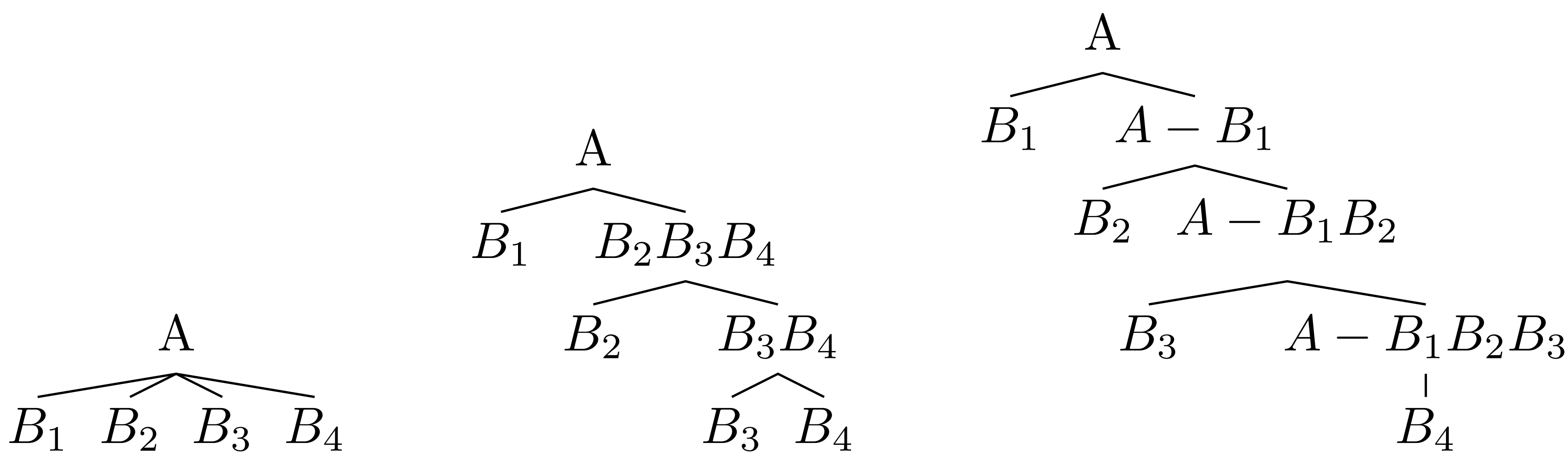


Left-factored Head-factored Right-factored

(assuming $H = B_2$)

◊◊More on binarization

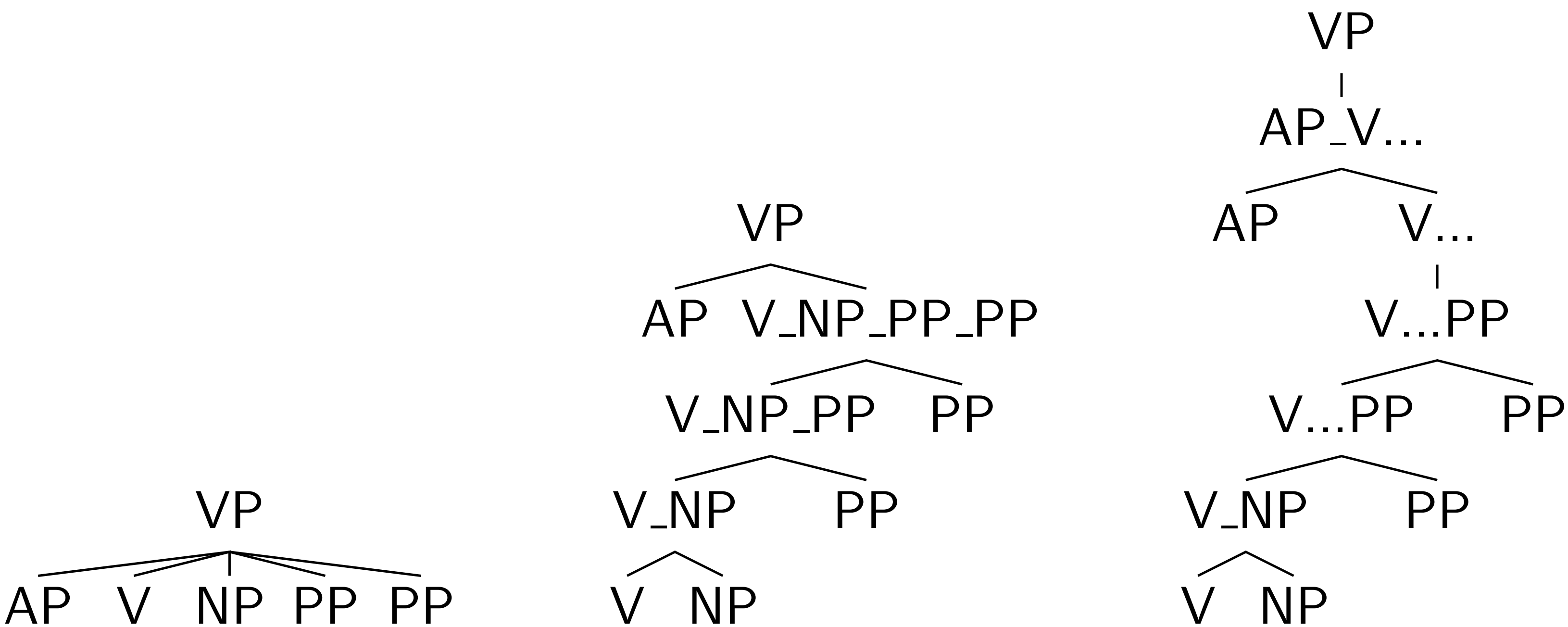
- Binarization usually produces large numbers of new nonterminals
- These all appear in a certain position (e.g., end of production)
- *Design your parser loops and indexing so this is maximally efficient*
- Top-down and left-corner parsing benefit from specially designed binarization that *delays choice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

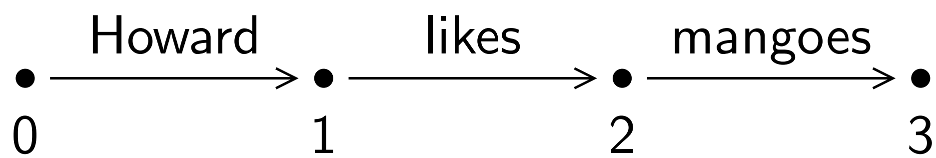
String positions

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

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

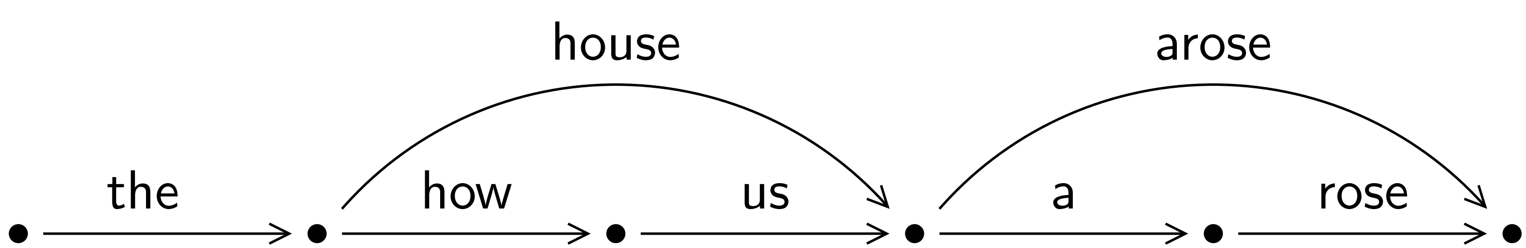
A substring of w is represented by a pair (i, j) of string positions, where $0 \leq i \leq j \leq n$.

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



Example: $w_{0,1} =$ Howard, $w_{1,3} =$ 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 $A \to B\ C$ or $A \to x$, 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: $\mathrm{P}(A \Rightarrow^\star w_{i-1,i}) \;=\; p(A \to w_{i-1,i})$ for $i = 1, \ldots, n$

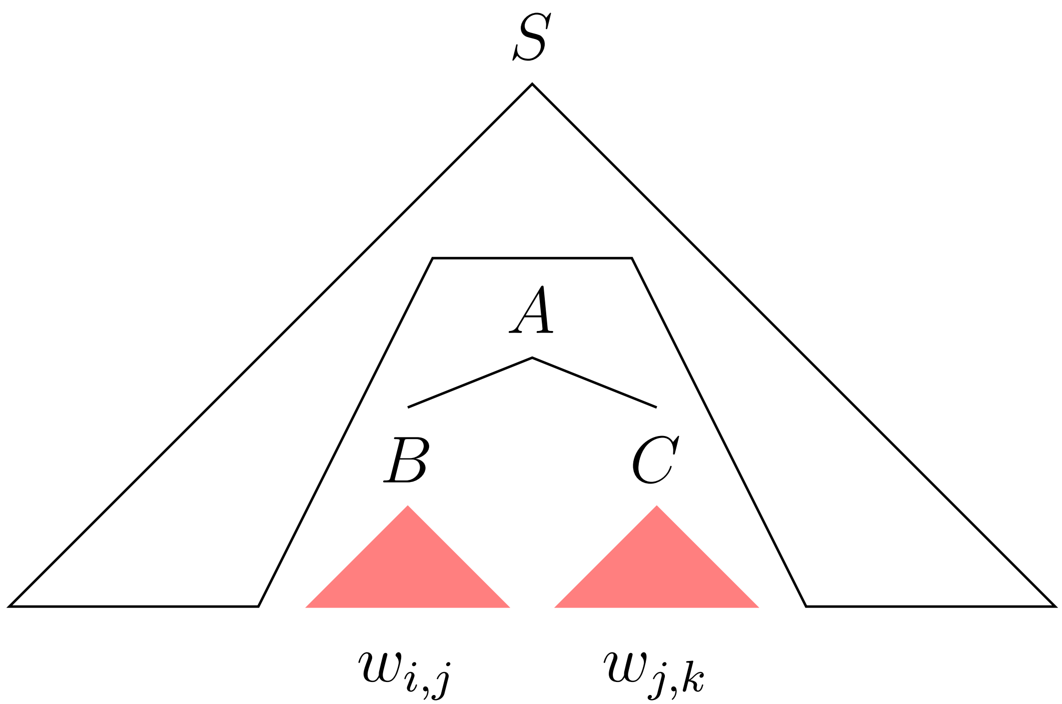
Recursion: $\mathrm{P}(A \Rightarrow^\star w_{i,k})$

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

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

Dynamic programming recursion

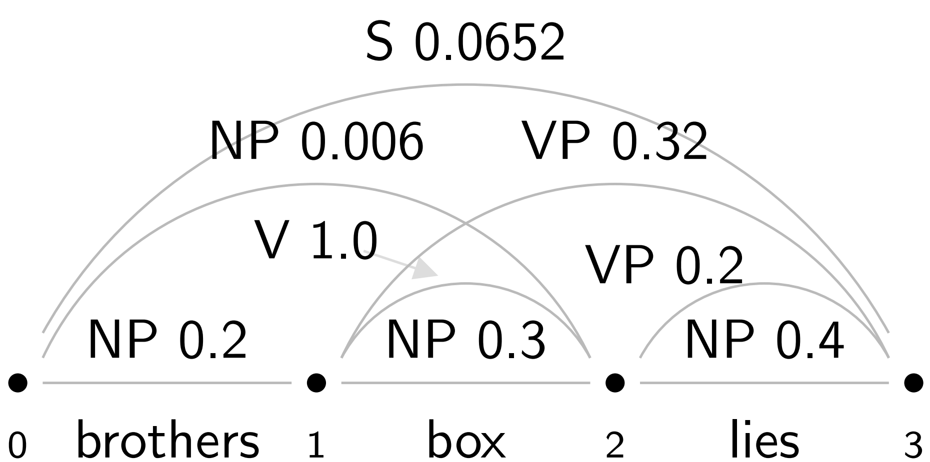
$$
\begin{aligned}
&\mathrm{P}_G(A \Rightarrow^* w_{i,k}) \\
&\quad = \sum_{j=i+1}^{k-1} \sum_{A \,\longrightarrow\, B\,C \in \mathcal{R}(A)} p(A \rightarrow B\,C) \mathrm{P}_G(B \Rightarrow^* w_{i,j}) \mathrm{P}_G(C \Rightarrow^* w_{j,k})
\end{aligned}
$$



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

Example PCFG parse

1.0	S → NP VP	0.1	NP → NP NP
0.2	NP → brothers	0.3	NP → box
0.4	NP → lies	1.0	V → box
0.8	VP → V NP	0.2	VP → lies

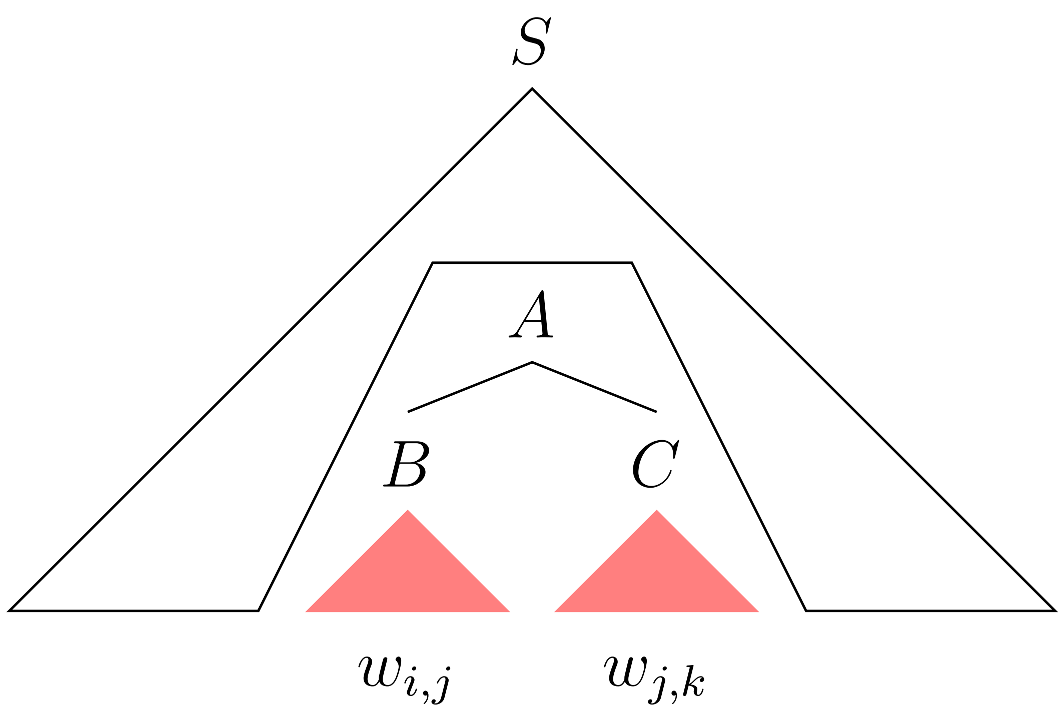


	1	2	3
0	NP 0.2	NP 0.006	S 0.0652
1		NP 0.3 V 1.0	VP 0.32
2			NP 0.4 VP 0.2

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

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

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



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

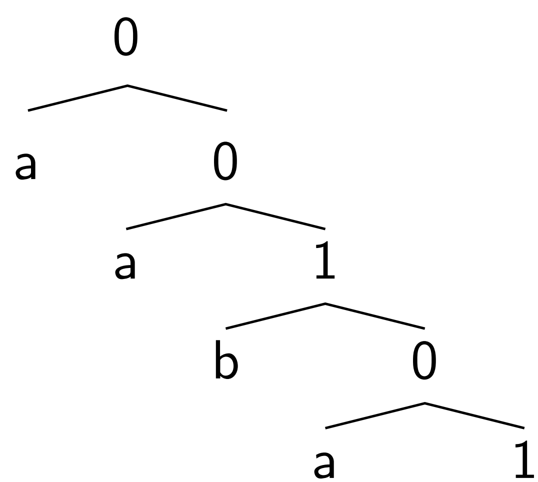
Because FSA trees are *uniformly right branching*,

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

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

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

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



◇◇Unary productions and unary closure

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

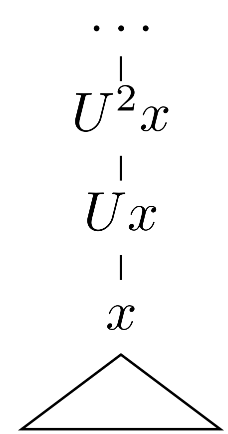
The *unary closure matrix* is $C_{ij} = \mathrm{P}(A_i \Rightarrow^{\star} A_j)$ for all $A_i, A_j \in \mathcal{S}$

Define $U_{ij} = p(A_i \rightarrow A_j)$ 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 cheap.

Finding the most likely parse of a string

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

(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* $\mathrm{P}_G(\widehat{\psi})$.

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

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

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:

$$\hat{p}_{\mathrm{MLE}} = \operatorname*{argmax}_p L_D(p) = \operatorname*{argmax}_p \ell_D(p)$$

$\diamond\diamond$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*

$\diamond\diamond$PCFG MLE from visible data

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

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

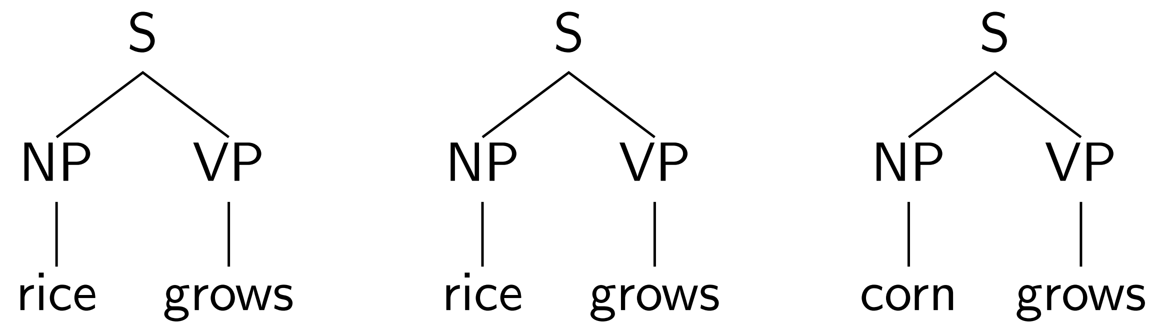
Introduce $|\mathcal{S}|$ Lagrange multipliers $\lambda_B, B \in \mathcal{S}$ for the constraints $\sum_{B\to\beta\in\mathcal{R}(B)} p(B \to \beta) = 1$. Then:

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

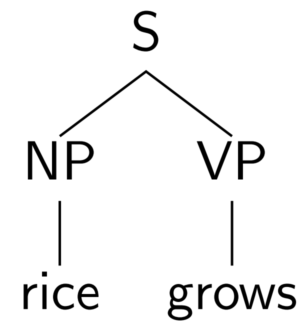
Setting this to 0, $\quad p(A \to \alpha) \;=\; \dfrac{n_{A\to\alpha}(D)}{\sum_{A\to\alpha'\in\mathcal{R}(A)} n_{A\to\alpha'}(D)}$

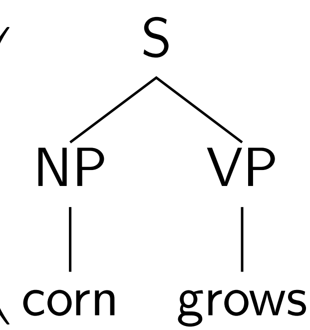
So the MLE for PCFGs is the *relative frequency estimator*

Example: Estimating PCFGs from visible data



Rule	Count	Rel Freq
S $\rightarrow$ NP VP	3	1
NP $\rightarrow$ rice	2	$2/3$
NP $\rightarrow$ corn	1	$1/3$
VP $\rightarrow$ 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 EM algorithm finds the MLE $\hat{p}(x) = L_x(p)$ of the *visible data* x.

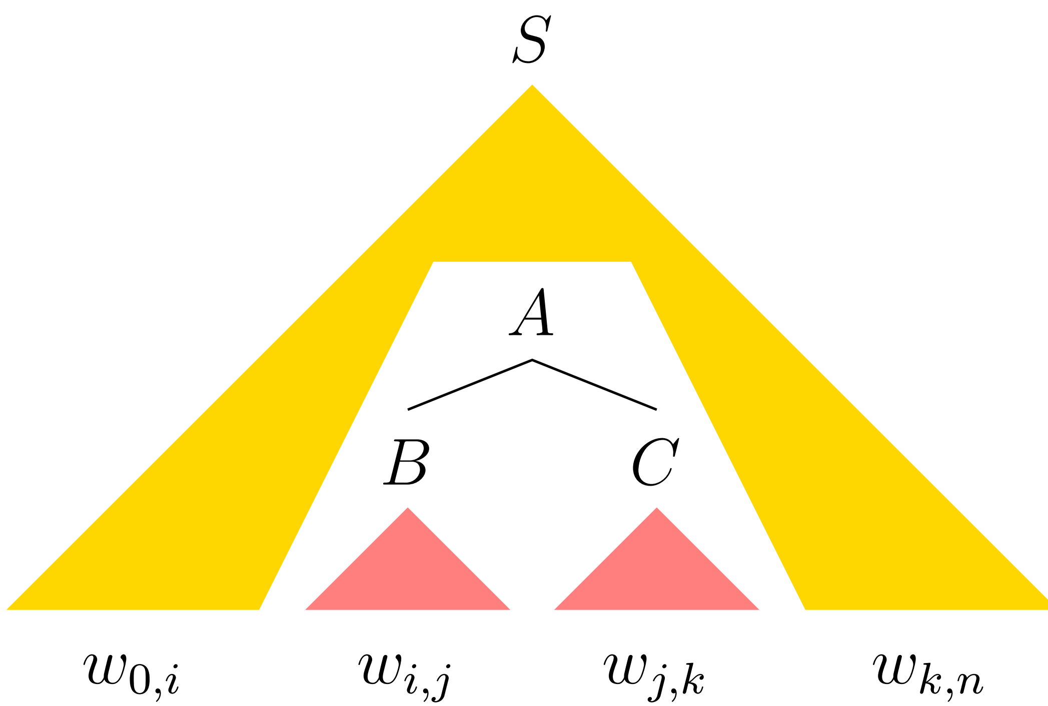
Sometimes it is not necessary to explicitly generate the pseudo-data (x, y); often it is possible to perform the maximization step directly from *sufficient statistics* (for PCFGs, the expected production frequencies)

Dynamic programming for $E_G[n_{A \to B\, C}|w]$

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

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

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

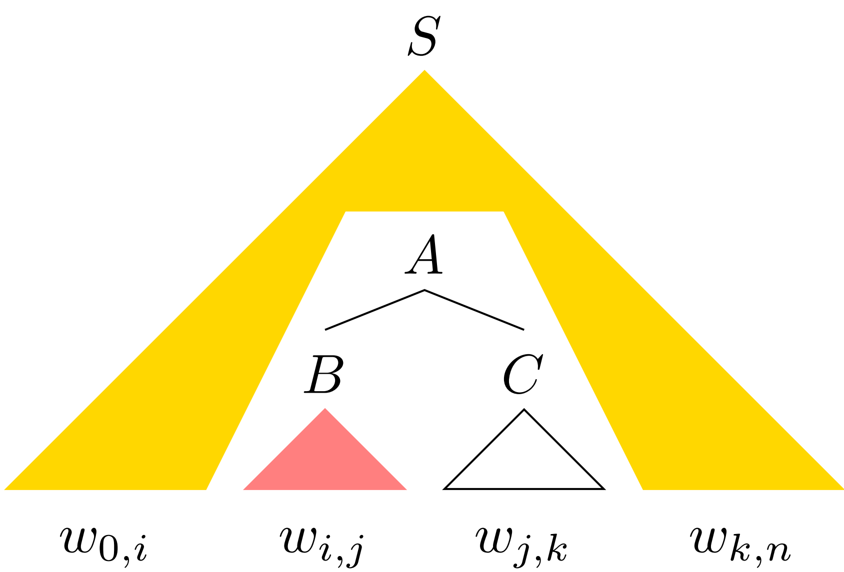
$$+ \sum_{l=k+1}^{n} \sum_{\substack{A,D \in \mathcal{S} \\ A \to C\, D \in \mathcal{R}}} \mathrm{P}(S \Rightarrow^* w_{0,j}\, A\, w_{l,n}) p(A \to C\, D) \mathrm{P}(D \Rightarrow^* w_{k,l})$$

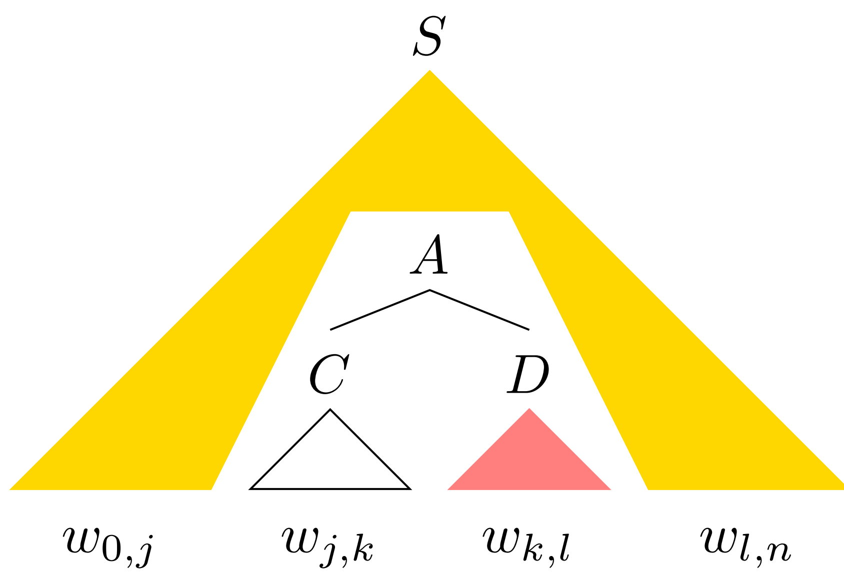
Recursion in $\mathrm{P}_G(S \Rightarrow^* w_{0,i}\, A\, w_{k,n})$

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

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

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





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 → V	0.2
VP → V NP	0.2
VP → NP V	0.2
VP → V NP NP	0.2
VP → NP NP V	0.2
...	...
Det → the	0.1
N → the	0.1
V → the	0.1

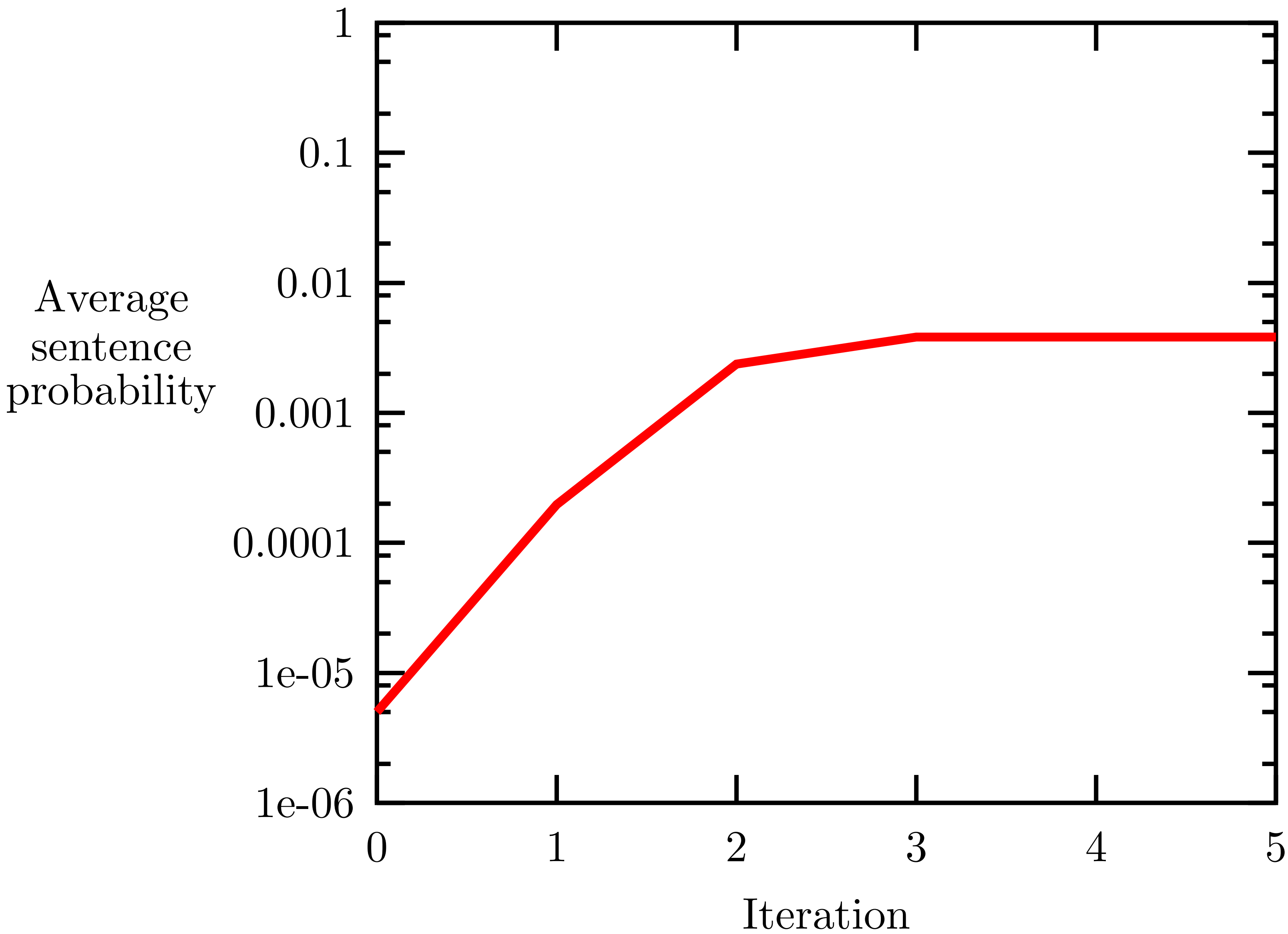
"English" input

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

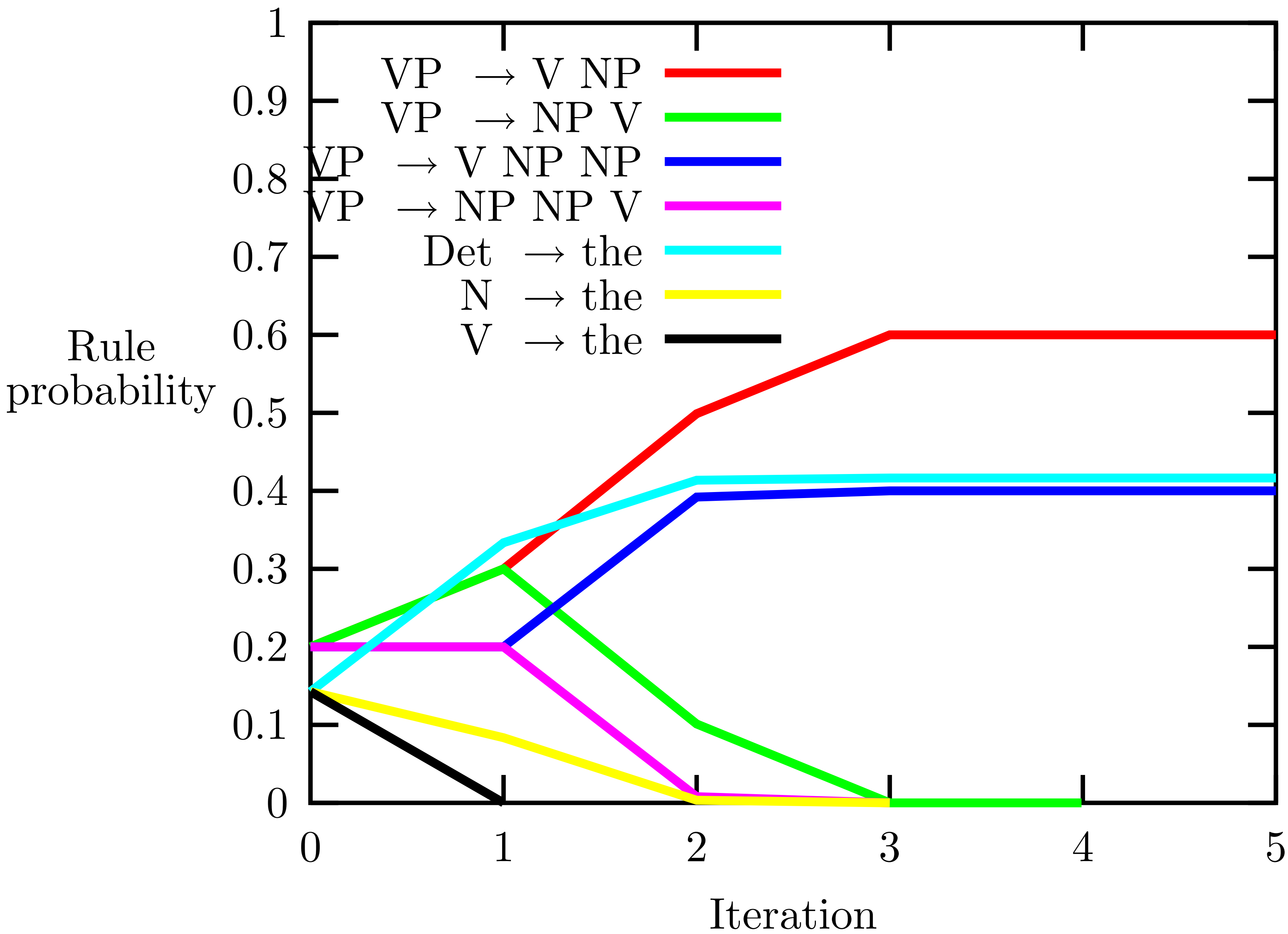
"pseudo-Japanese" input

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

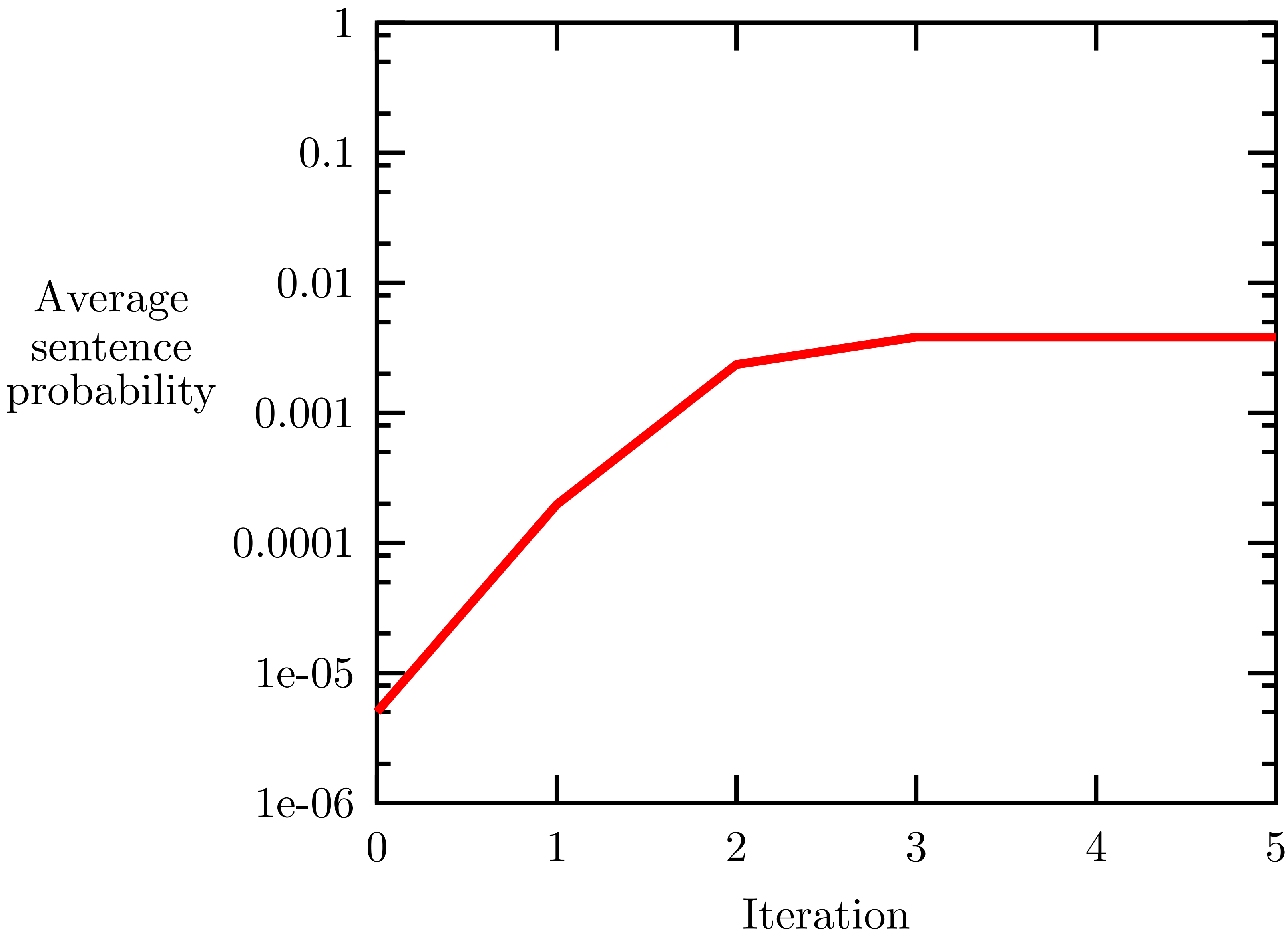
Probability of "English"



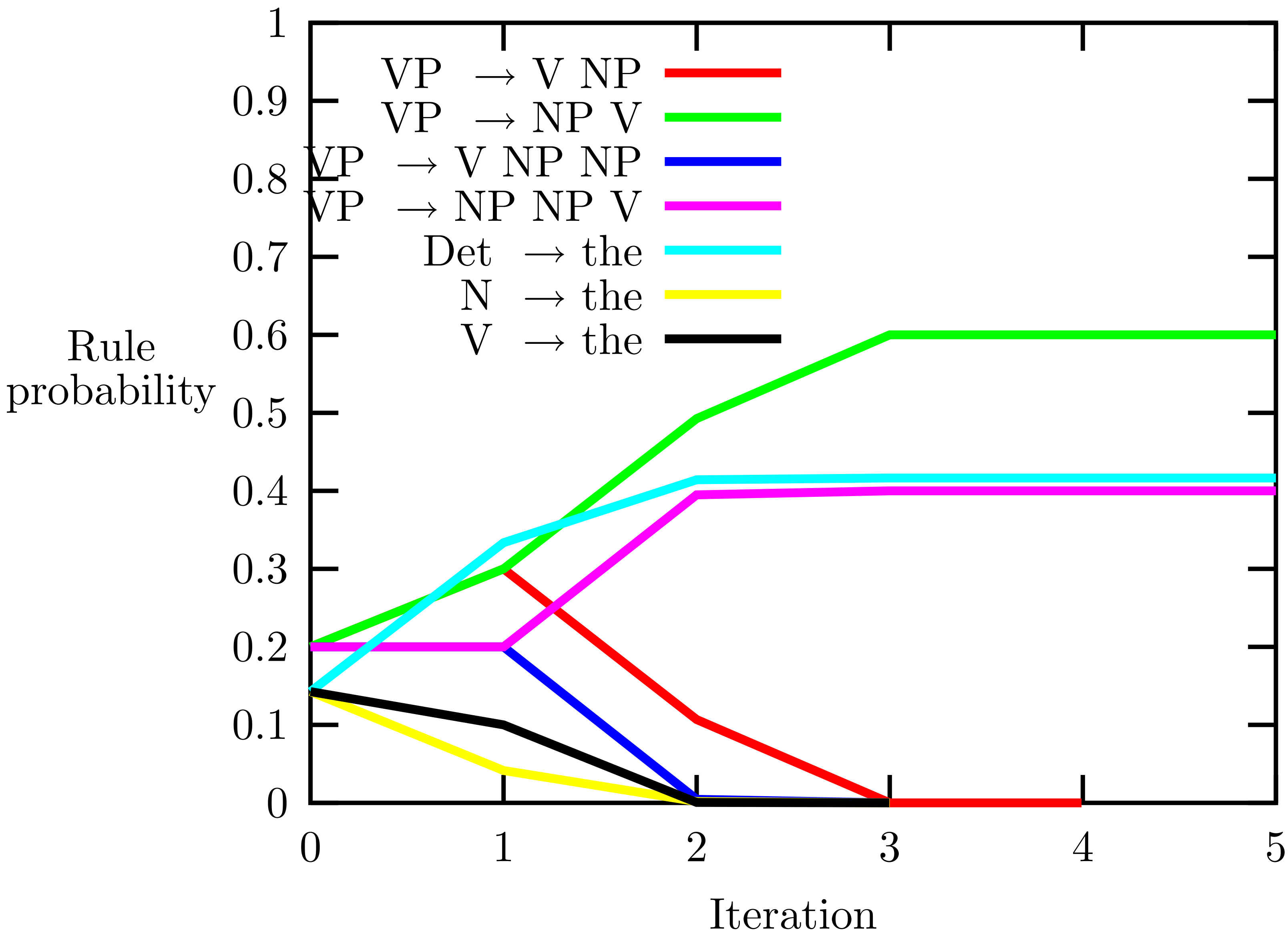
Rule probabilities from "English"



Probability of "Japanese"



Rule probabilities from "Japanese"

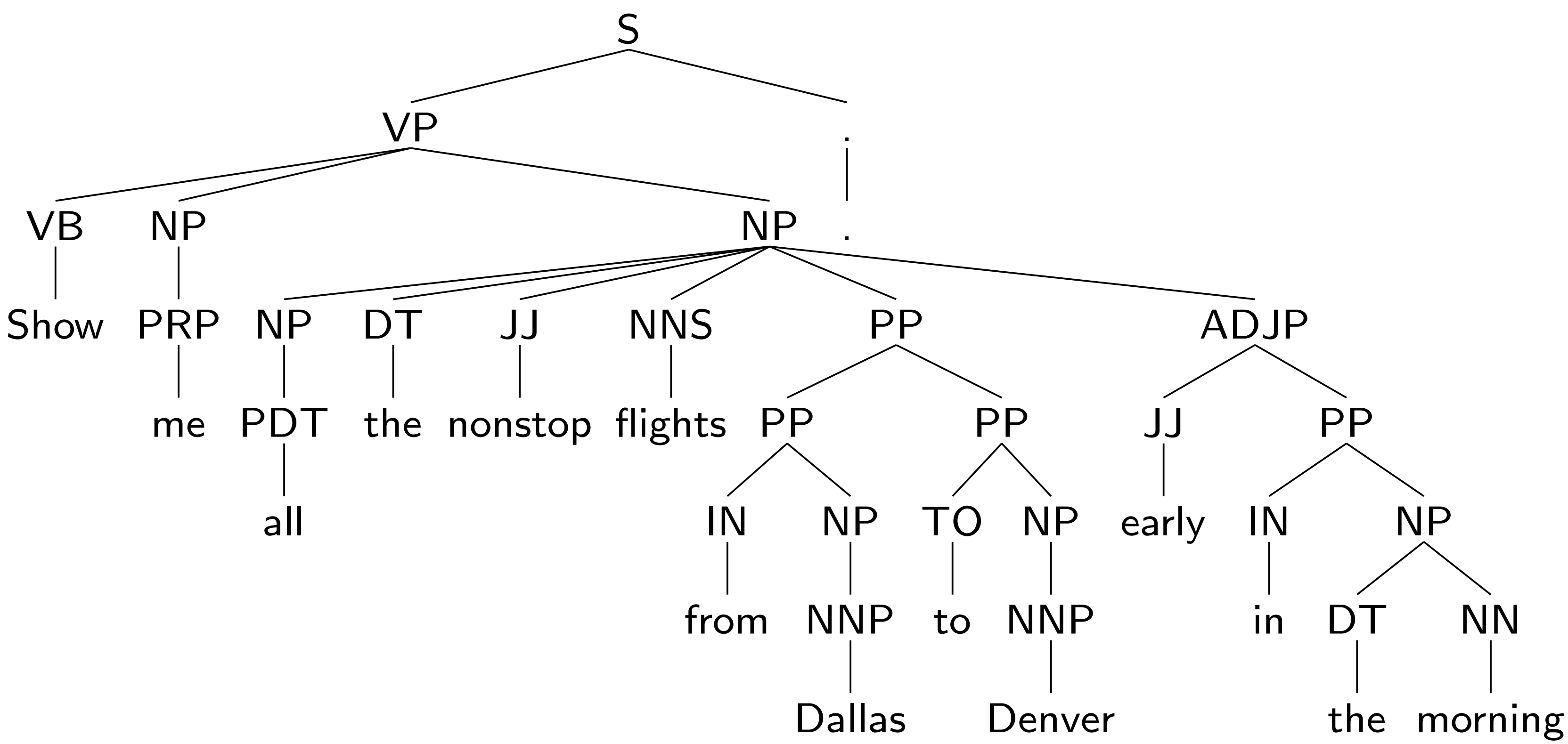


Learning in statistical paradigm

- 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

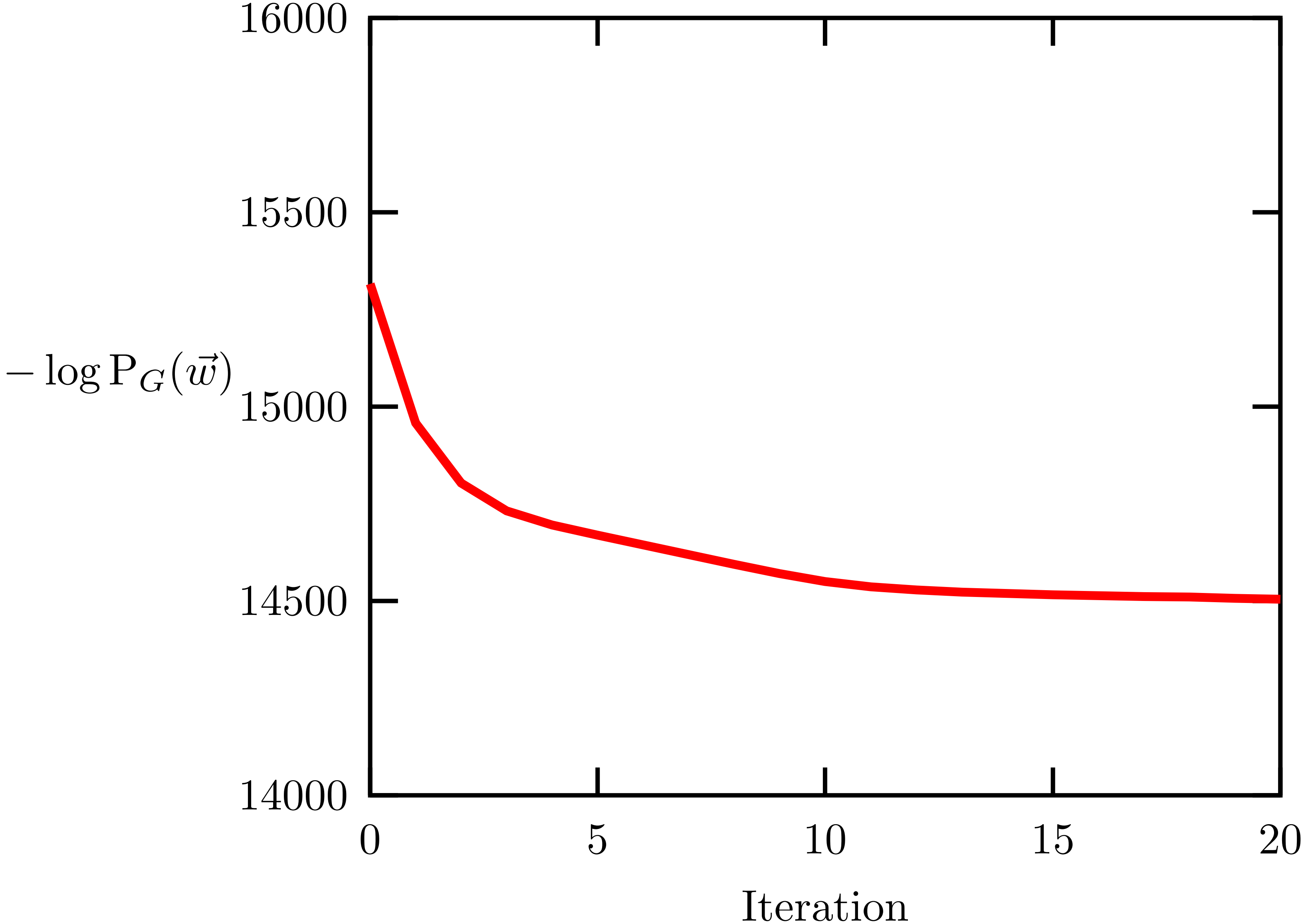
3. Apply EM (to strings alone) to re-estimate production probabilities.

4. At each iteration:

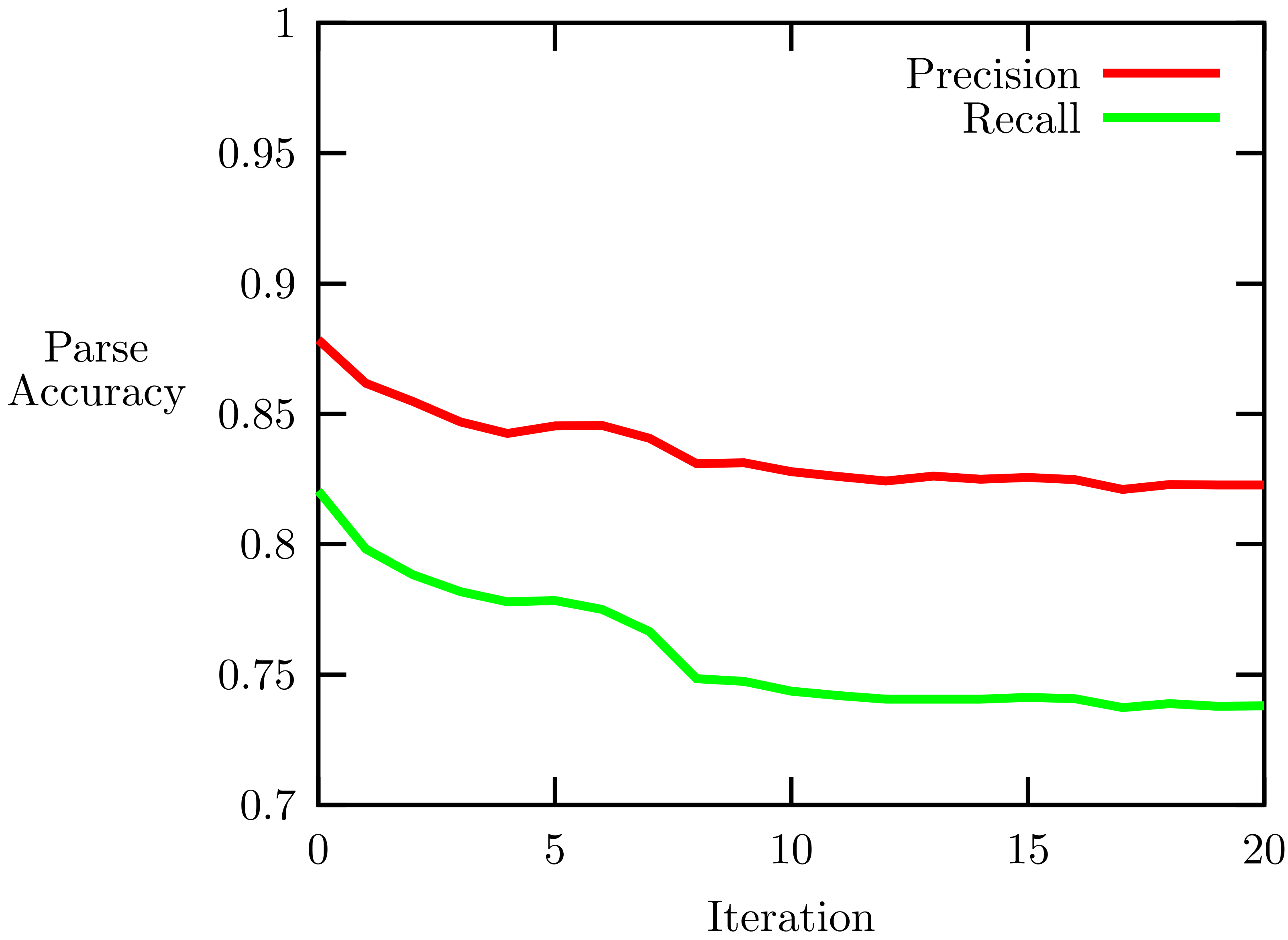
   - Measure the likelihood of the training data and the quality of the parses produced by each grammar.

   - 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?

- EM assigns trees to strings to maximize the marginal probability of the strings, but the trees weren't designed with that in mind
- We have an "intended interpretation" of categories like NP, VP, etc., which EM has no way of knowing
- Our grammar models are defective; real languages aren't context-free
- How can information about $\mathrm{P}(w)$ provide information about $\mathrm{P}(\psi|w)$?
- ... but no one really knows.

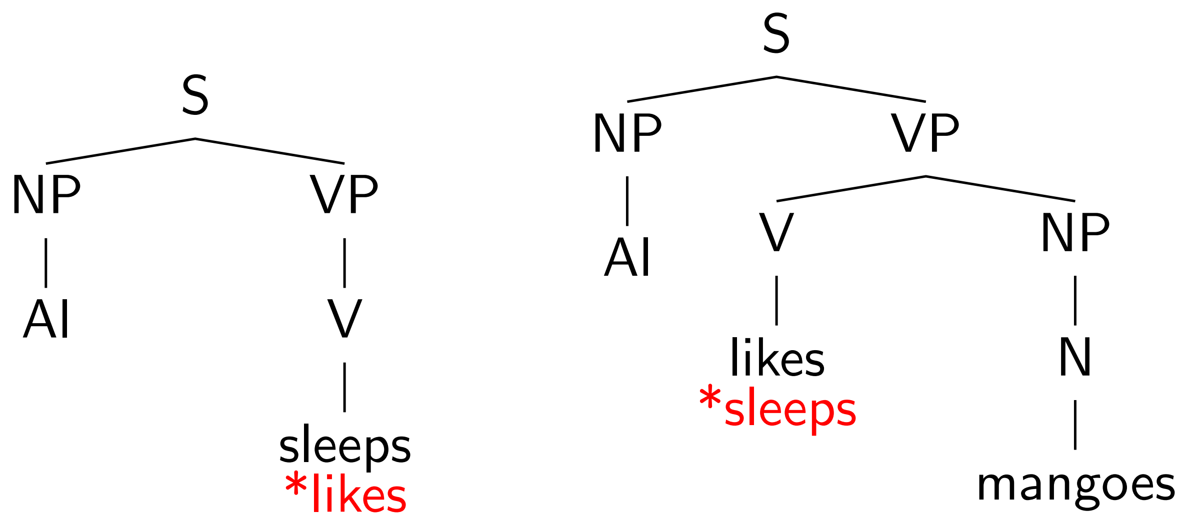
Topics

- 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.

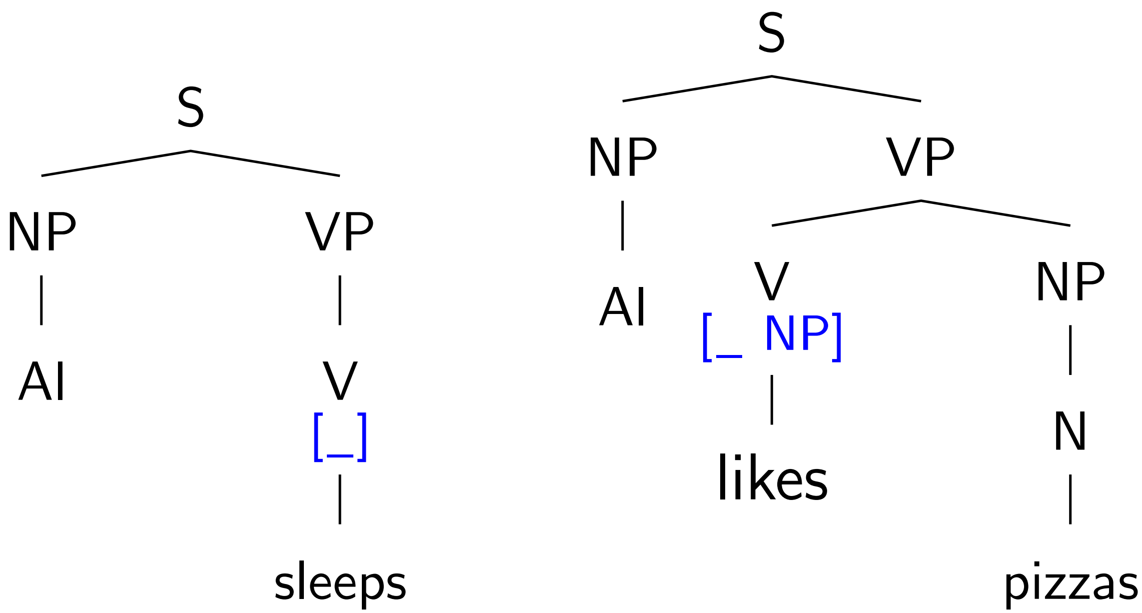
$R_3 = \{\mathsf{VP} \to \mathsf{V}, \mathsf{VP} \to \mathsf{V\,NP}, \mathsf{V} \to \mathsf{sleeps}, \mathsf{V} \to \mathsf{likes}, \ldots\}$



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.

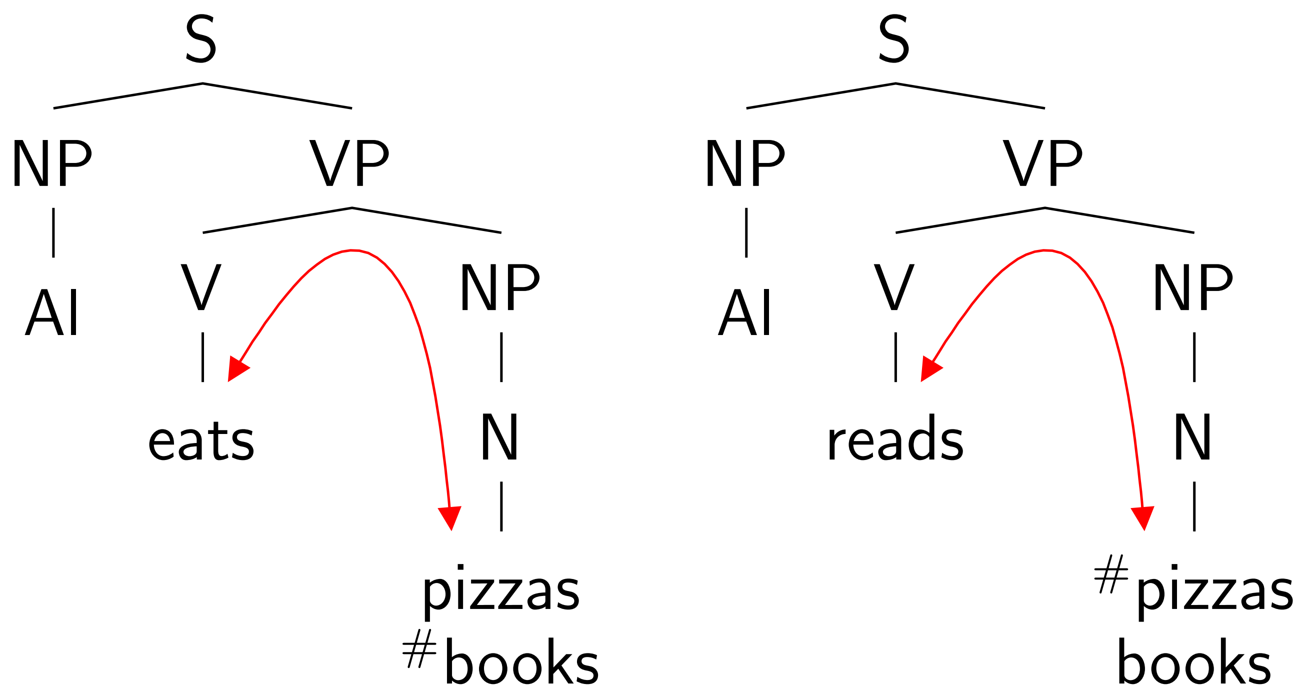


$$R_4 = \{\mathsf{VP} \rightarrow \underset{[\_]}{\mathsf{V}}, \mathsf{VP} \rightarrow \underset{[\_\ \mathsf{NP}]}{\mathsf{V}}\ \mathsf{NP}, \underset{[\_]}{\mathsf{V}} \rightarrow \mathsf{sleeps}, \underset{[\_\ \mathsf{NP}]}{\mathsf{V}} \rightarrow \mathsf{likes}, \ldots\}$$

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

Selectional preferences

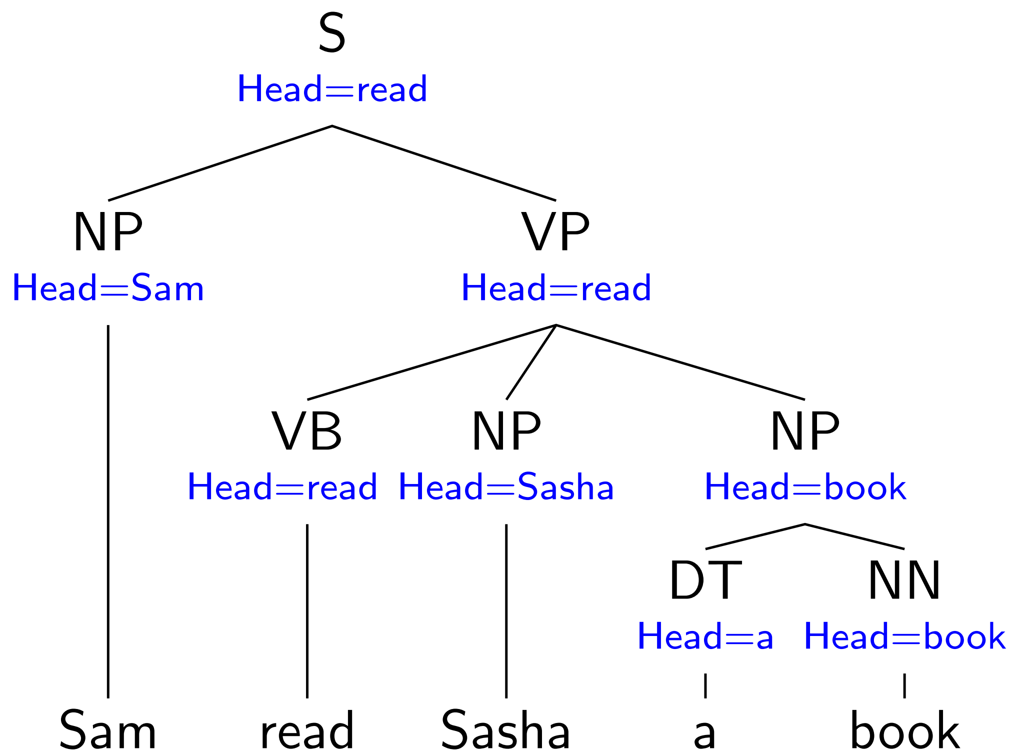
Head-to-head dependencies are an approximation to real-world knowledge.



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

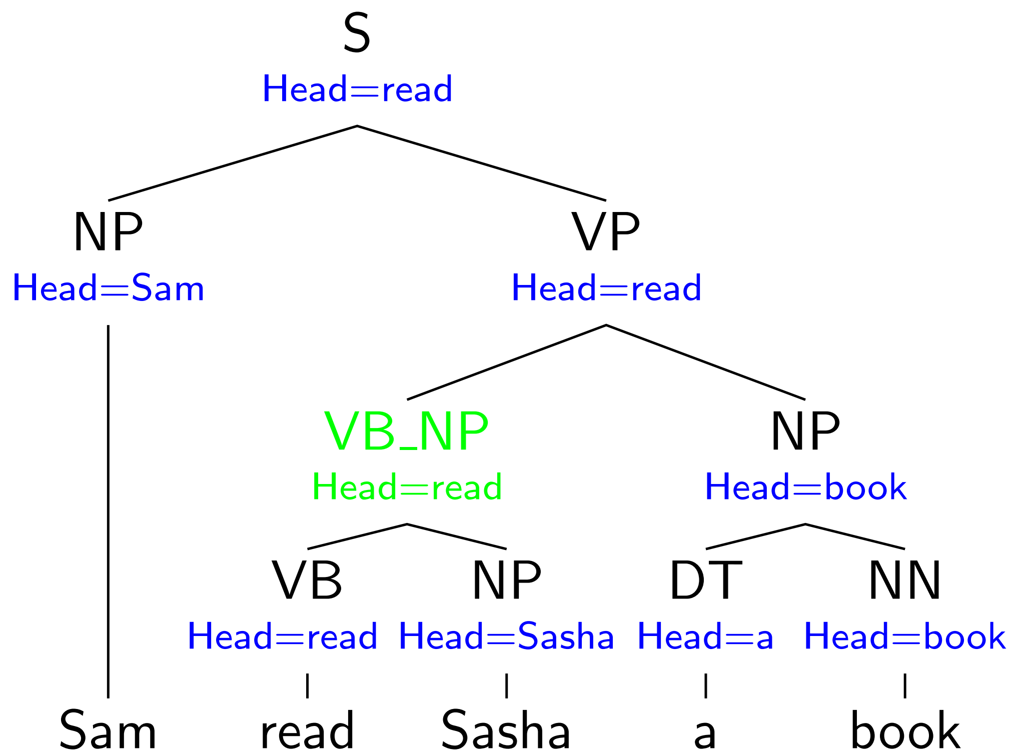
Al drives a (#toy model) car

Head to head dependencies



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

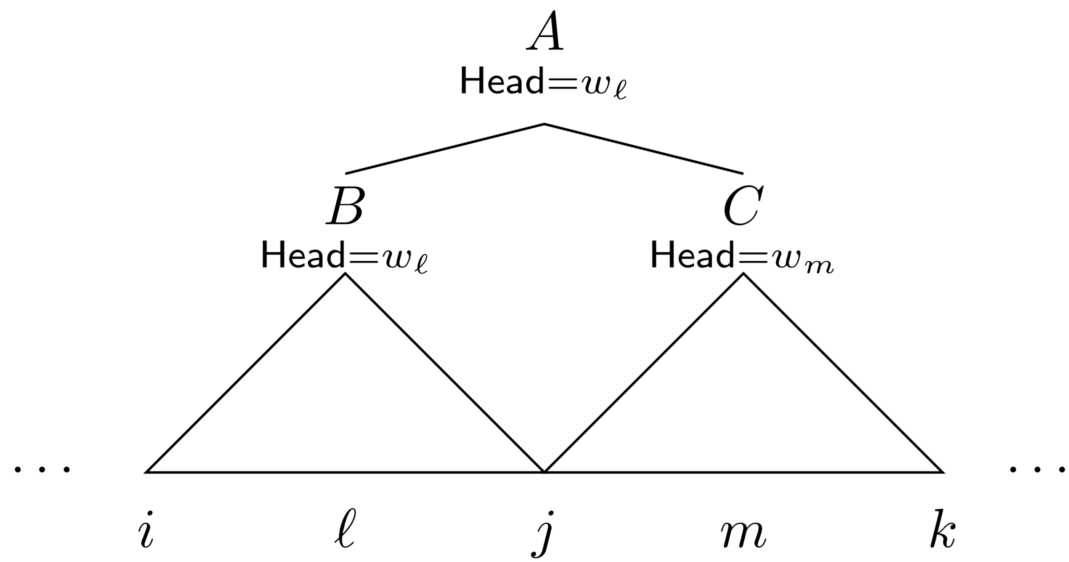
Binarization helps sparse data



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

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

Bi-lexical CFG parsing takes n^5 time

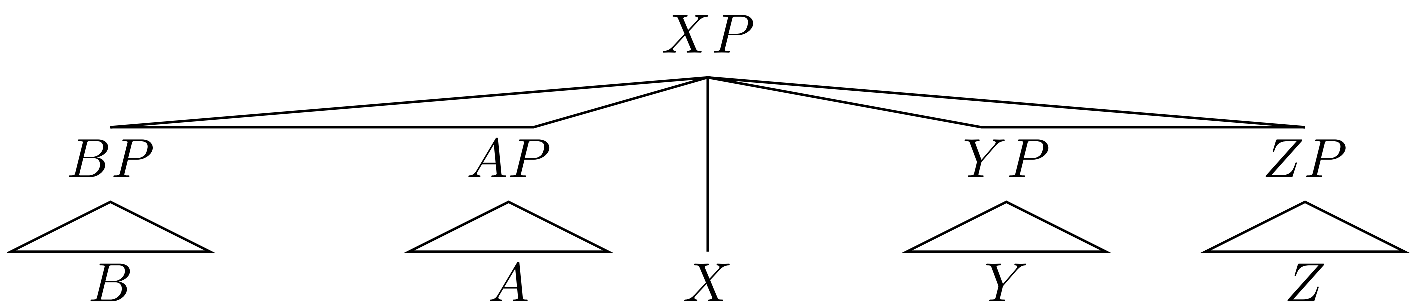


There are three string positions at the edges of constituents, plus two for the locations of the heads

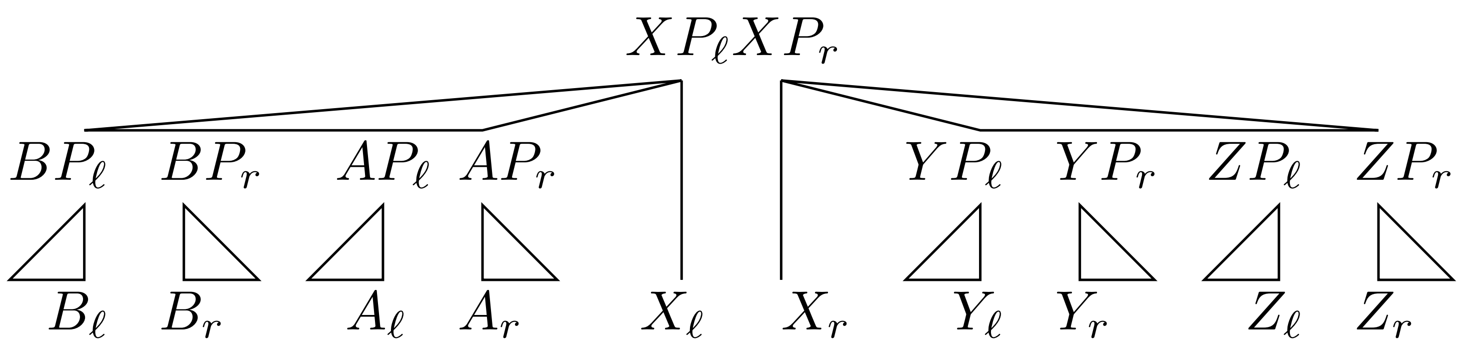
- in the worst case, bilexical parsing takes $|n|^5$ time
- the worst case arises when exhaustive parsing

Eisner and Satta's idea: change the grammar so that the heads are at the constituent edges

◊◊Eisner and Satta's bilexical parsing model



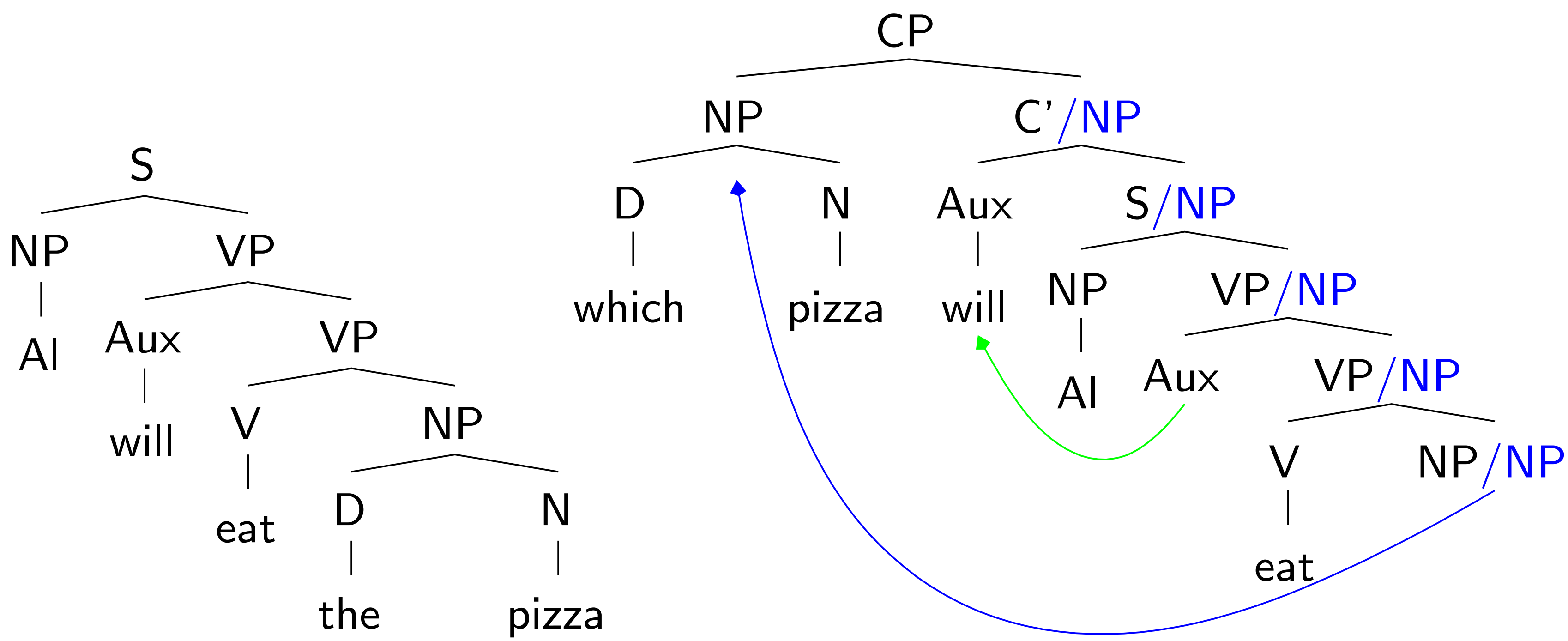
Split each node (including each word) into a left and a right half



Right factor the left halves and left factor the right halves

Synchronize the left and right halves by splitting the nonterminal states

Nonlocal “movement” dependencies

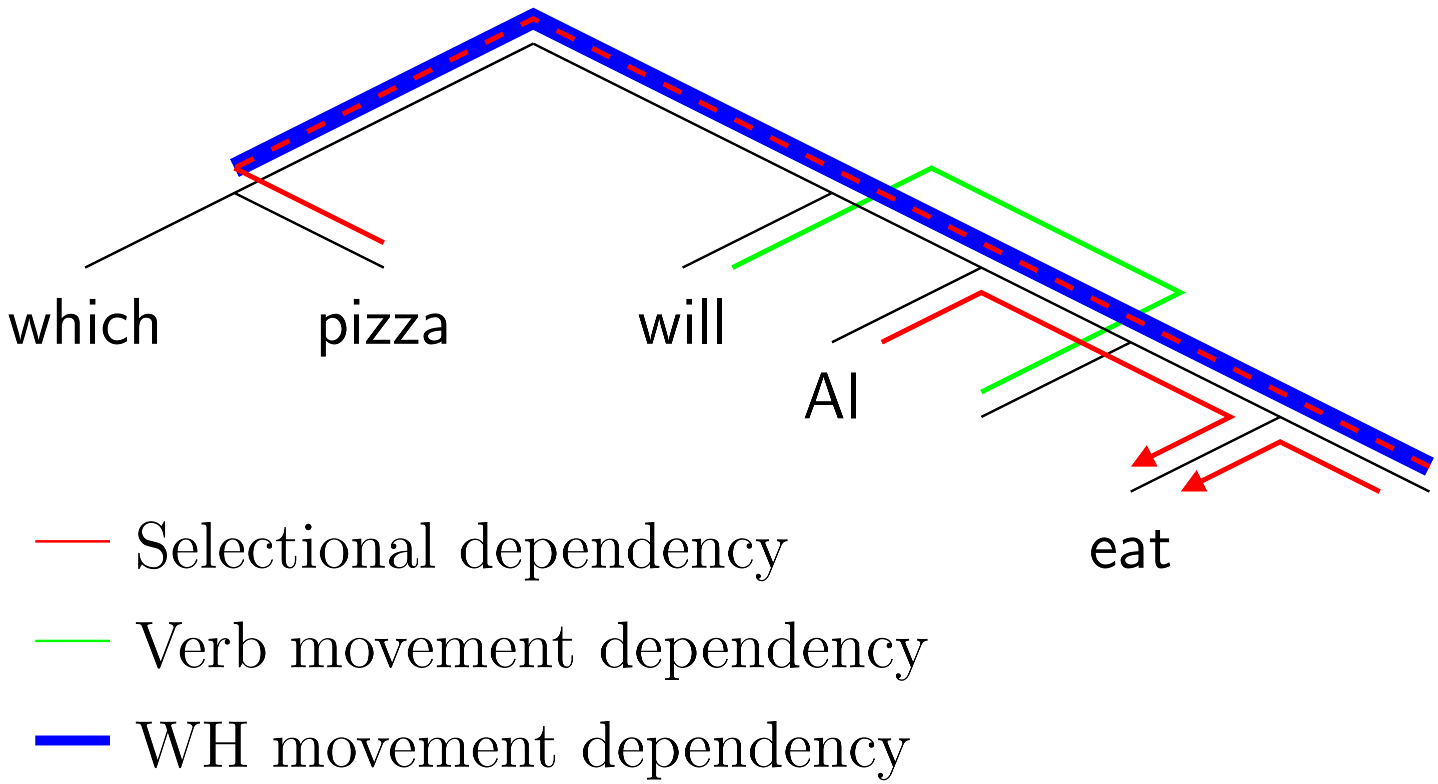


Subcategorization and selectional preferences are *preserved* under movement.

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

Structured nonterminals

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



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