An introduction to grammars and parsing

Mark Johnson

May, 2005

Talk outline

- What is computational linguistics?
- Probabilistic grammars
- Identifying phrase structure (parsing)
- Learning phrase structure
- More realistic grammars

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

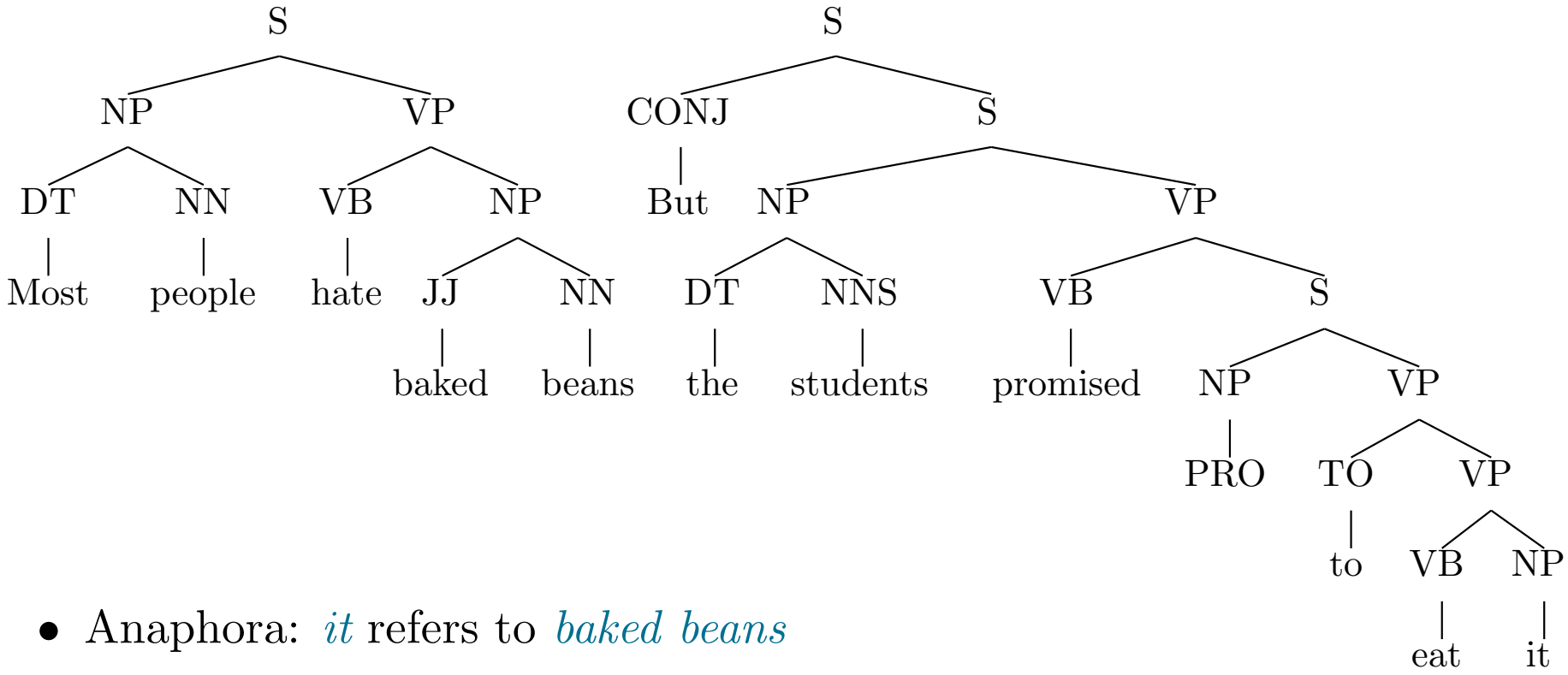
– We know little about neural networks that might support linguistic computations

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.

Talk outline

- What is computational linguistics?
- Probabilistic grammars
- Identifying phrase structure (parsing)
- Learning phrase structure
- More realistic grammars

Context free grammars

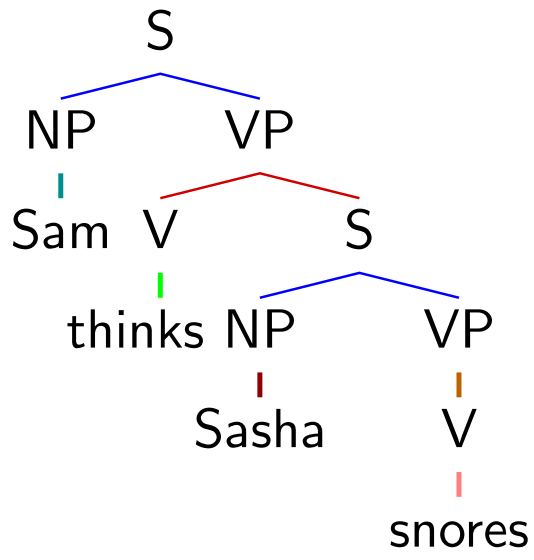
A *context-free grammar* $G = (\mathcal{V}, \mathcal{S}, s, \mathcal{R})$ consists of:

- $\mathcal{V}$, 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 $A \rightarrow X_1 \ldots X_n$, where $A \in \mathcal{S}$ and each $X_i \in \mathcal{S} \cup \mathcal{V}$
- $s \in \mathcal{S}$ is called the *start symbol* ($s_0 = \mathsf{S}$)

G *generates* a tree ψ iff

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

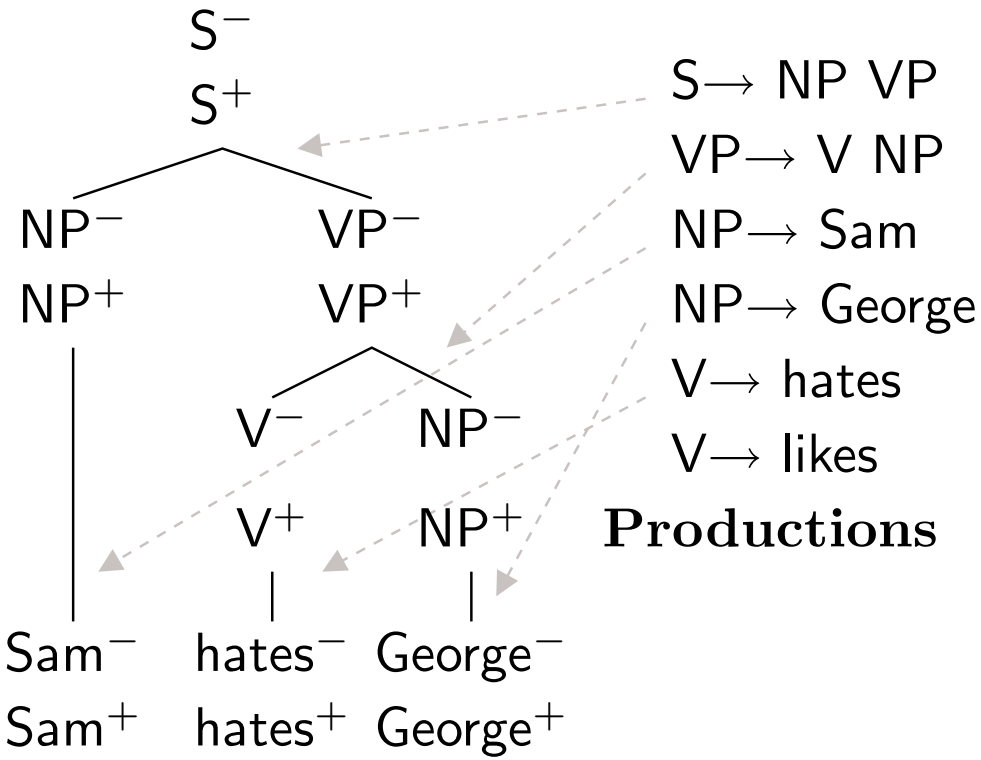
G generates a string $w \in \mathcal{V}^\star$ iff w is the *terminal yield* of a tree generated by G

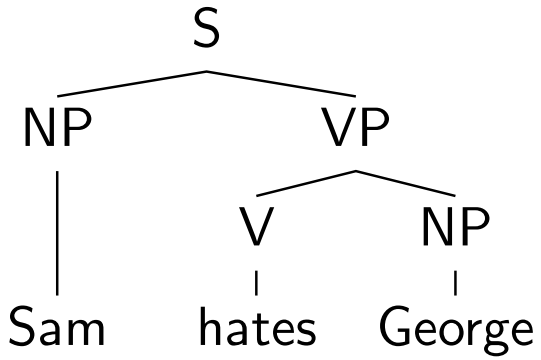


Productions

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

CFGs as "plugging" systems



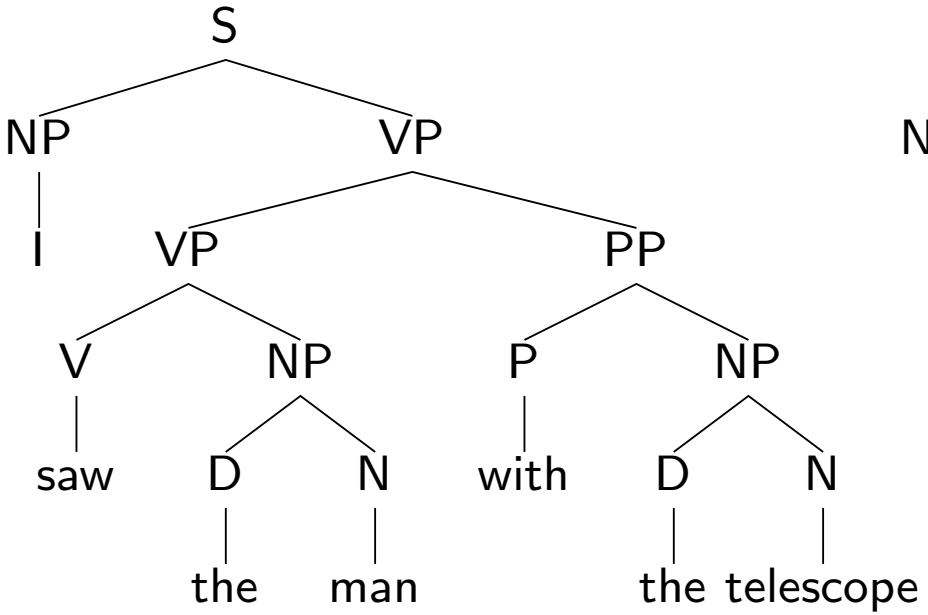


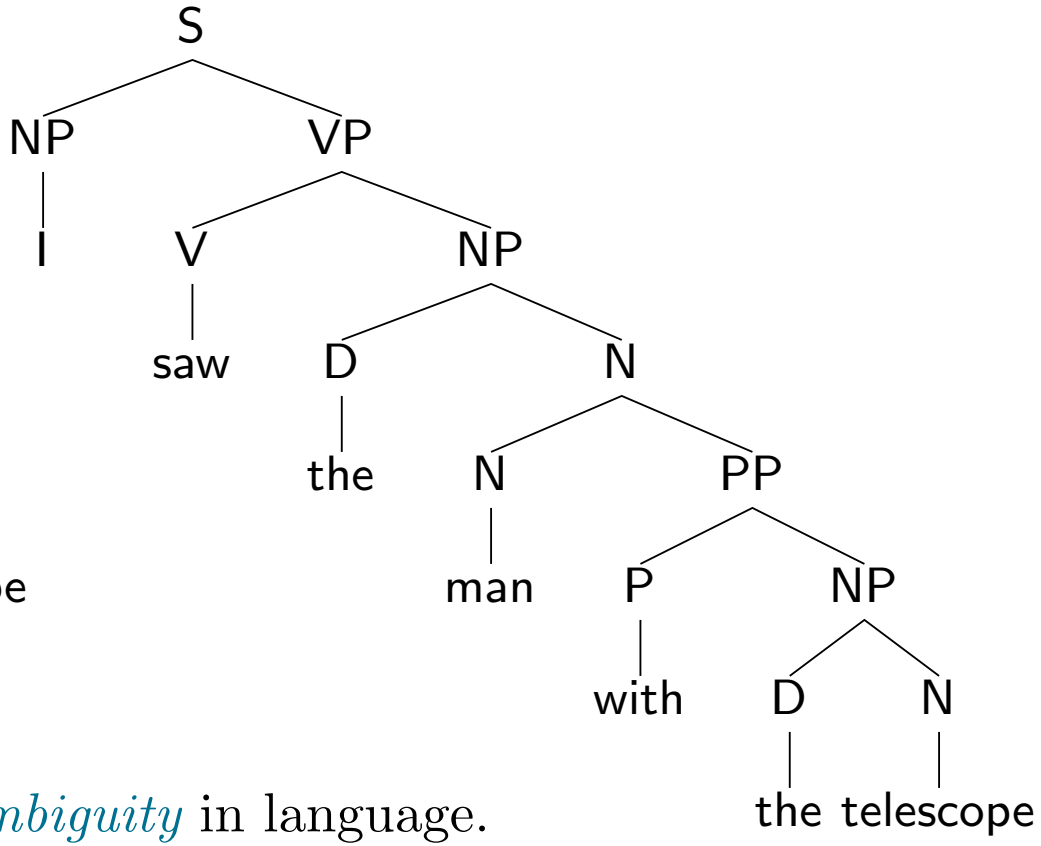
"Pluggings" **Resulting tree**

- 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

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





- 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 $s \Rightarrow_G \ldots \Rightarrow_G w$. $\Rightarrow_G^\star$ is the *reflexive and transitive closure* of $\Rightarrow_G$.

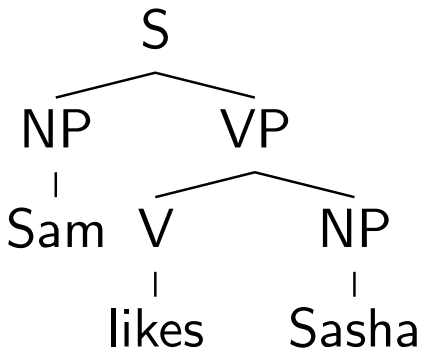
The *language generated* by G is $\{w : s \Rightarrow^\star w, w \in \mathcal{V}^\star\}$.

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

S
⇒ NP VP
⇒ NP V NP
⇒ Sam V NP
⇒ Sam V Sasha
⇒ Sam likes Sasha

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

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



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* $p(A \to \beta) = \mathrm{P}(\beta|A)$ for each $A \to \beta \in \mathcal{R}$, the conditional probability of an A expanding to β

A production $A \to \beta$ 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 $\gamma, \delta \in (\mathcal{V} \cup \mathcal{S})^\star$ and $w \in \mathcal{V}^\star$.

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

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

Talk outline

- What is computational linguistics?
- Probabilistic grammars
- Identifying phrase structure (parsing)
- Learning phrase structure
- More realistic grammars

Computing the probability of a tree

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

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

Things we want to compute

1. *What is the probability* $\mathrm{P}_G(w)$ *of the string* w*?* (language modeling)

$$\mathrm{P}_G(w) \quad = \quad \mathrm{P}_G(s \Rightarrow^* w) \quad = \quad \sum_{\psi \in \Psi_G(w)} \mathrm{P}_G(\psi)$$

2. *What is the most probable parse* $\hat{\psi}(w)$ *of a string* w*?* (parsing)

$$\hat{\psi}(w) \quad = \quad \operatorname*{argmax}_{\psi \in \Psi_G(w)} \mathrm{P}_G(\psi)$$

where:

$\Psi_G(w)$ is the set of parse trees for w generated by G, and
$\mathrm{P}_G(\psi)$ is the probability of tree ψ wrt grammar G.

Naive algorithm:

1. Compute set of parse trees $\Psi_G(w)$ for w
2. Take max/sum as appropriate

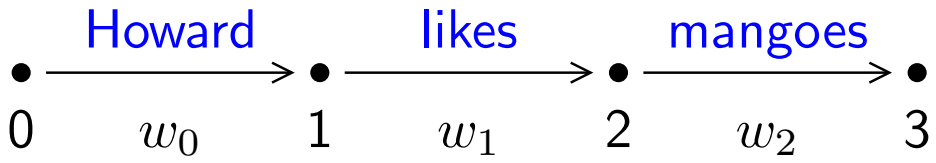
String positions

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

A *string position* of a string $w = x_0 \dots x_{n-1}$ is an integer $0 \leq i \leq n$.

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

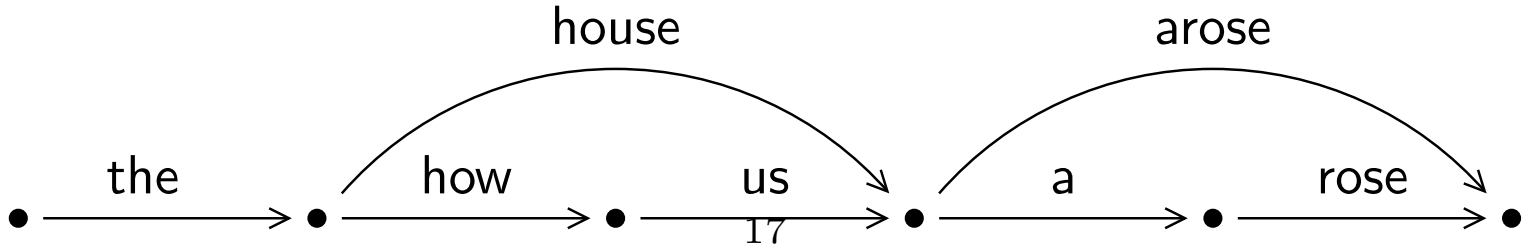
$w_{i,j}$ represents the substring $w_i \dots w_{j-1}$



Example:

$w_{0,1} =$ Howard, $w_{1,3} =$ likes mangoes, $w_{0,0} = w_{1,1} = w_{2,2} = w_{3,3} = \epsilon$

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



PCFG "Inside" algorithm

The *inside algorithm* for computing the probability of a string is a generalization of the backward algorithm.

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: $\mathrm{P}(A \Rightarrow^\star w_{i,i+1}) = p(A \to w_i)$ for $i = 0, \ldots, n-1$

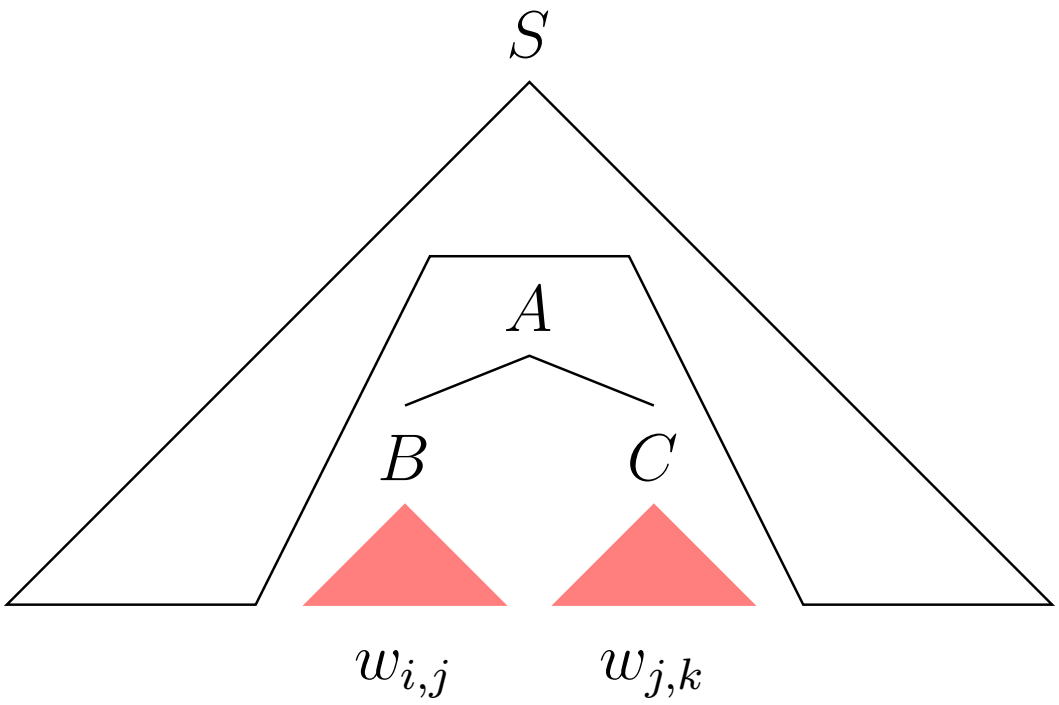
Recursion: $\mathrm{P}(A \Rightarrow^\star w_{i,k})$

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

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

Dynamic programming recursion

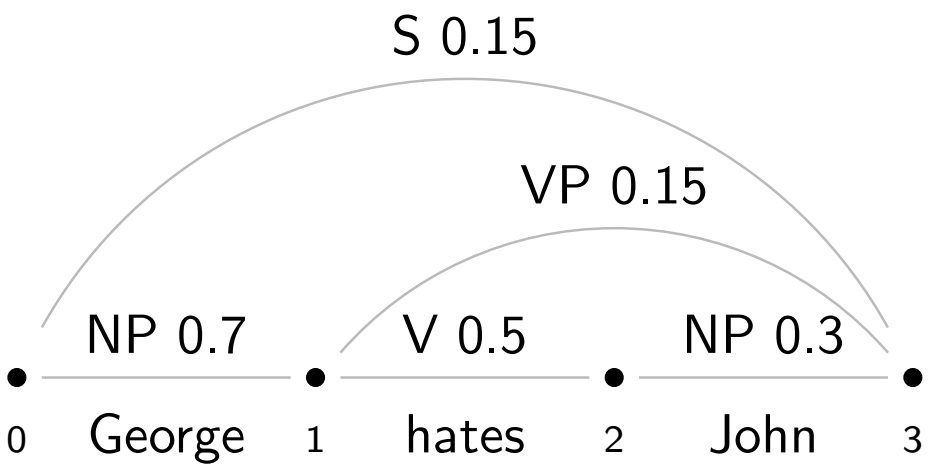
$$\mathrm{P}_G(A \Rightarrow^\star w_{i,k}) = \sum_{j=i+1}^{k-1} \sum_{A \to B\,C \in \mathcal{R}(A)} p(A \to B\,C)\mathrm{P}_G(B \Rightarrow^\star w_{i,j})\mathrm{P}_G(C \Rightarrow^\star w_{j,k})$$



$\mathrm{P}_G(A \Rightarrow^\star w_{i,k})$ 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 0.15
1		V 0.5	VP 0.15
2			NP 0.3

Talk outline

- What is computational linguistics?
- Probabilistic grammars
- Identifying phrase structure (parsing)
- Learning phrase structure
- More realistic grammars

Two approaches to computational linguistics

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

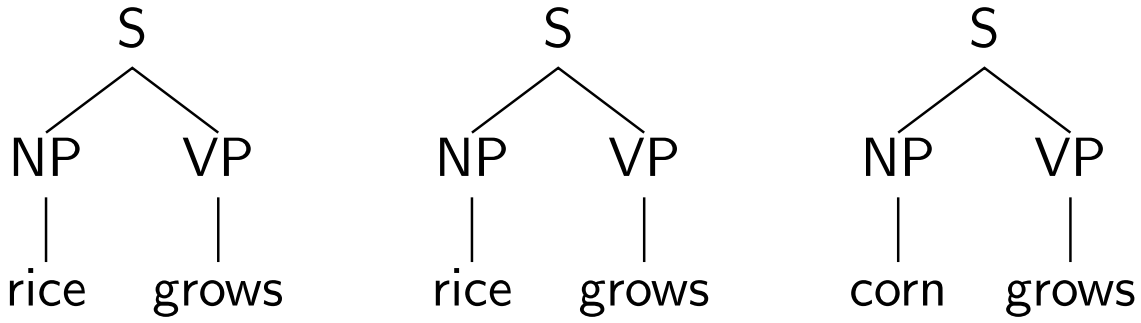
"Empiricist": Collect a corpus of examples, linguists annotate them with relevant information, a machine learning algorithm extracts generalizations

- 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*.

Estimating PCFGs from visible data



Rule	**Count**	**Rel Freq**
S → NP VP	3	1
NP → rice	2	2/3
NP → corn	1	1/3
VP → grows	3	1

$$\mathrm{P}\left(\begin{array}{c}\mathsf{S}\\ \mathsf{NP}\quad\mathsf{VP}\\ \mathsf{rice}\quad\mathsf{grows}\end{array}\right) = 2/3$$

$$\mathrm{P}\left(\begin{array}{c}\mathsf{S}\\ \mathsf{NP}\quad\mathsf{VP}\\ \mathsf{corn}\quad\mathsf{grows}\end{array}\right) = 1/3$$

(The relative frequency estimator is also the MLE for PFSA, of course).

Properties of MLE

The relative frequency estimator is the MLE for PCFGS, so it has the following properties:

- *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
- *Sparse data* is the big problem with the MLE.
  - Rules that do not occur in the training data get zero probability
- Aside from this, 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

Unsupervised training and EM

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 (e.g., rule probabilities)
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) has "frequency" $\mathrm{P}_{p_i}(y|x)$

   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.

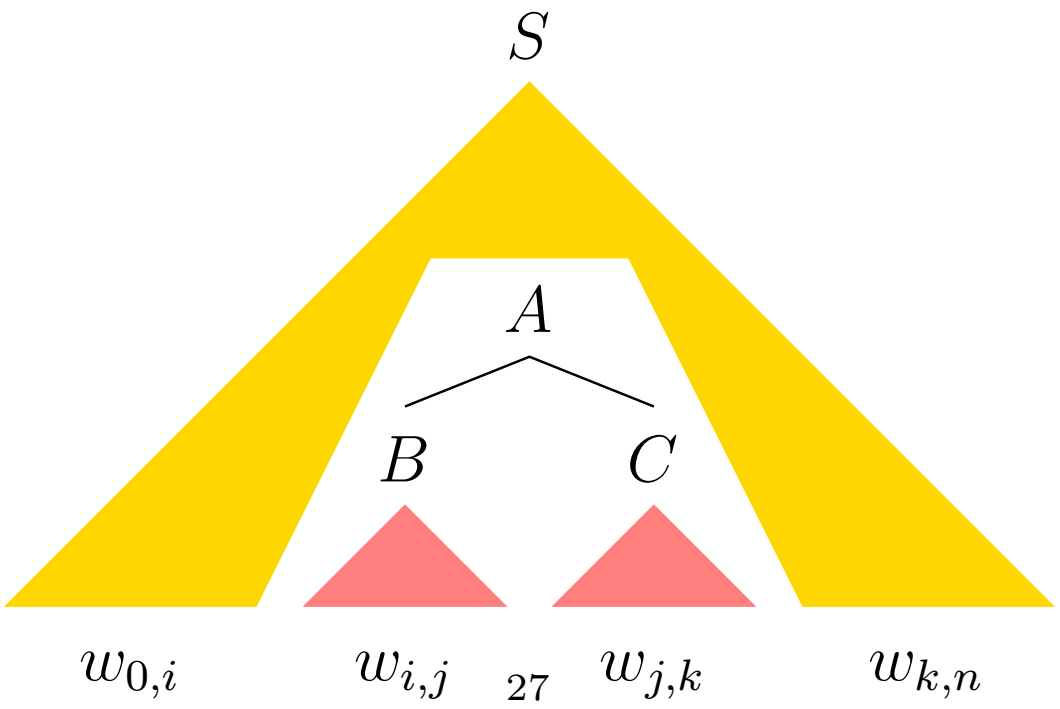
Sometimes it is not necessary to explicitly generate the pseudo-data (x, y); often it is possible to perform the maximization step directly from *sufficient statistics* (for PCFGs, the expected production frequencies)

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:

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



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

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

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

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

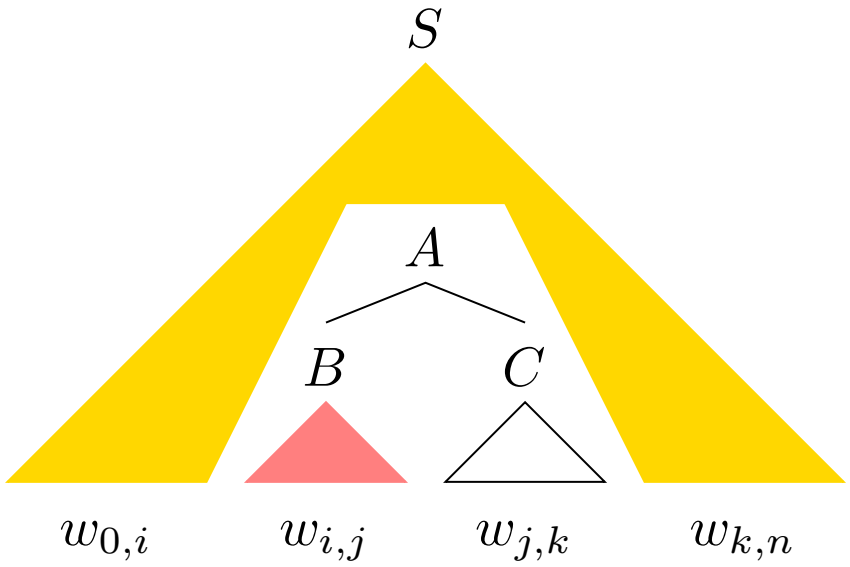
$$+ \sum_{l=k+1}^{n} \sum_{\substack{A,D \in \mathcal{S} \\ A \to C\, D \in \mathcal{R}}} \mathrm{P}(S \Rightarrow^\star w_{0,j}\, A\, w_{l,n}) p(A \to C\, D) \mathrm{P}(D \Rightarrow^\star w_{k,l})$$

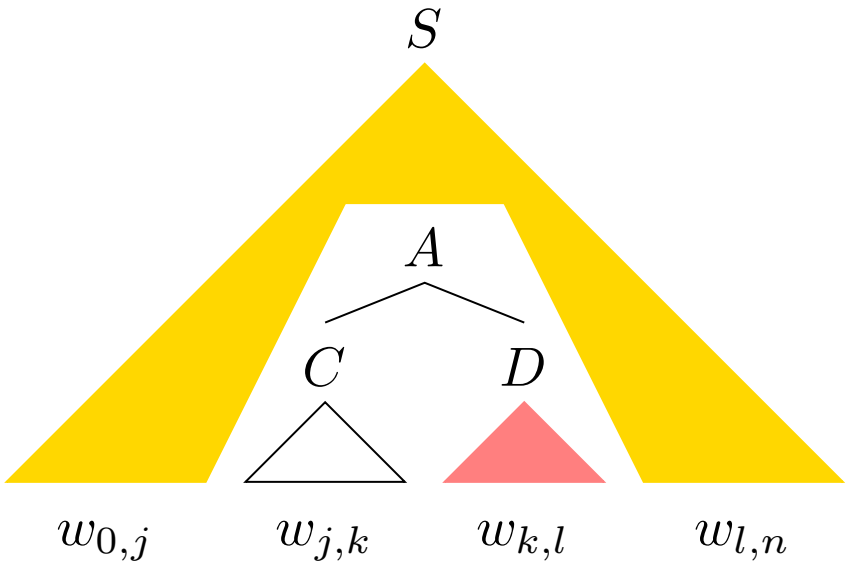
Recursion in $\mathrm{P}_G(S \Rightarrow^\star w_{0,i}\, A\, w_{k,n})$

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

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

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





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

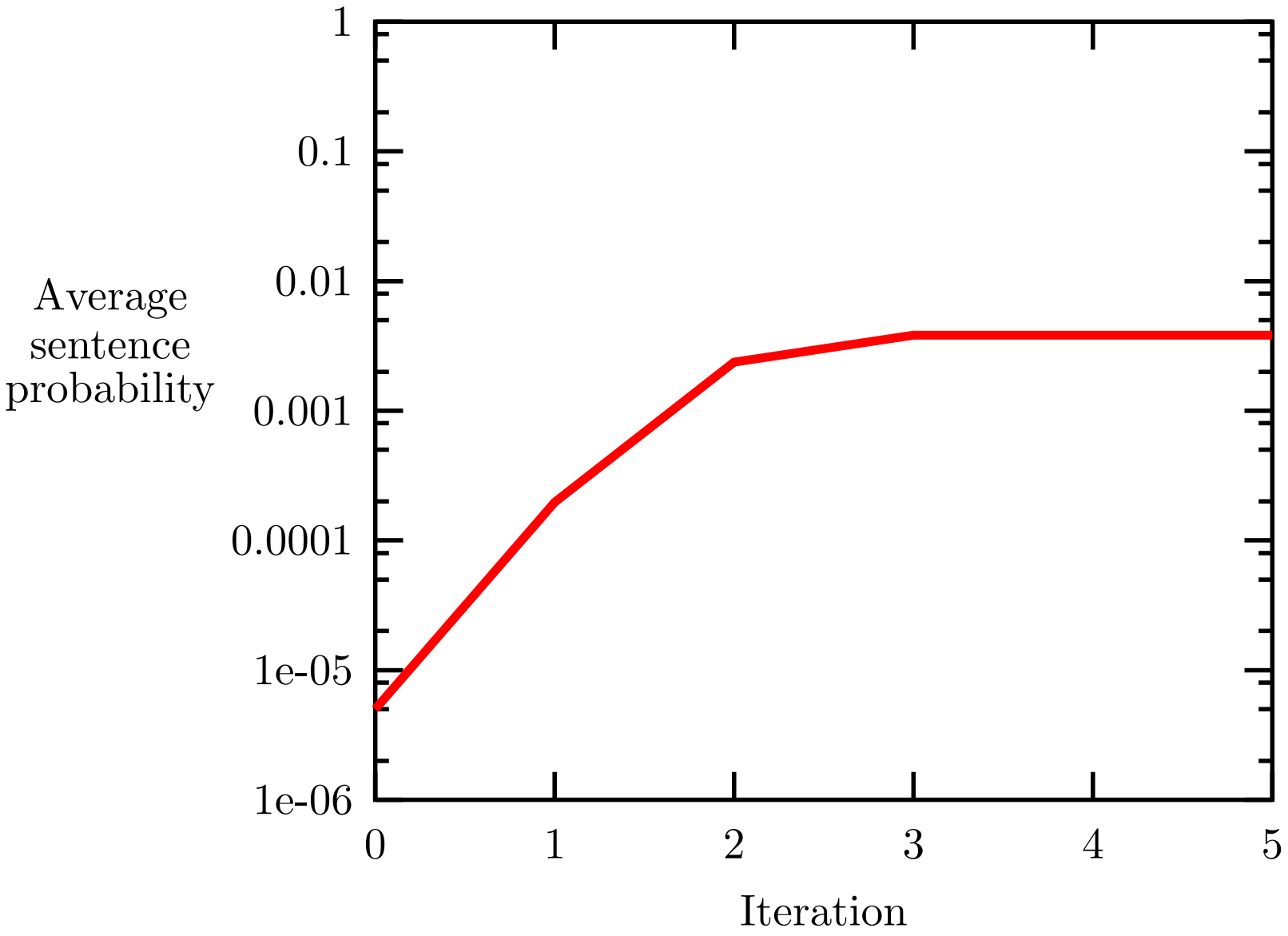
"English" input

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

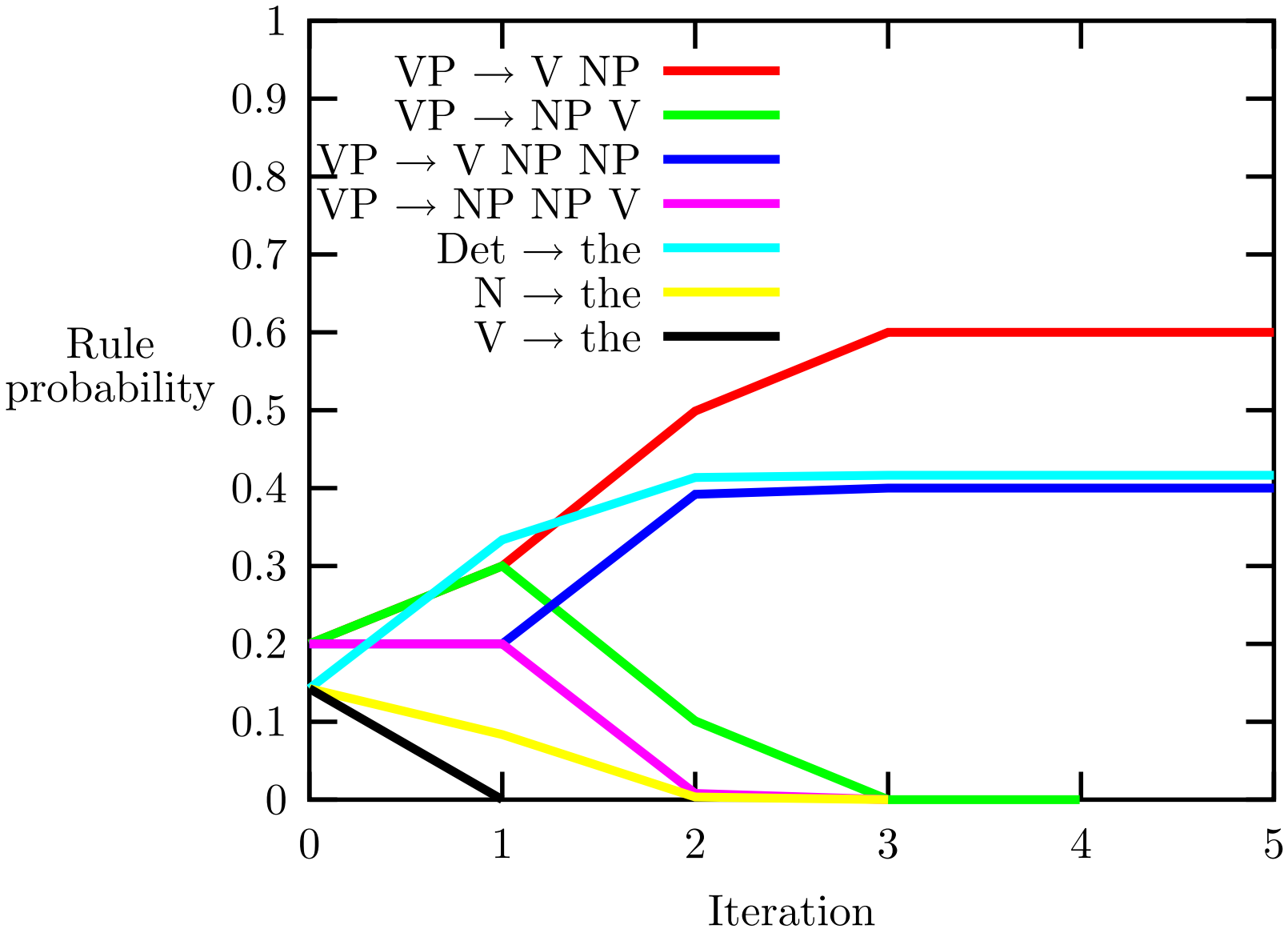
"pseudo-Japanese" input

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

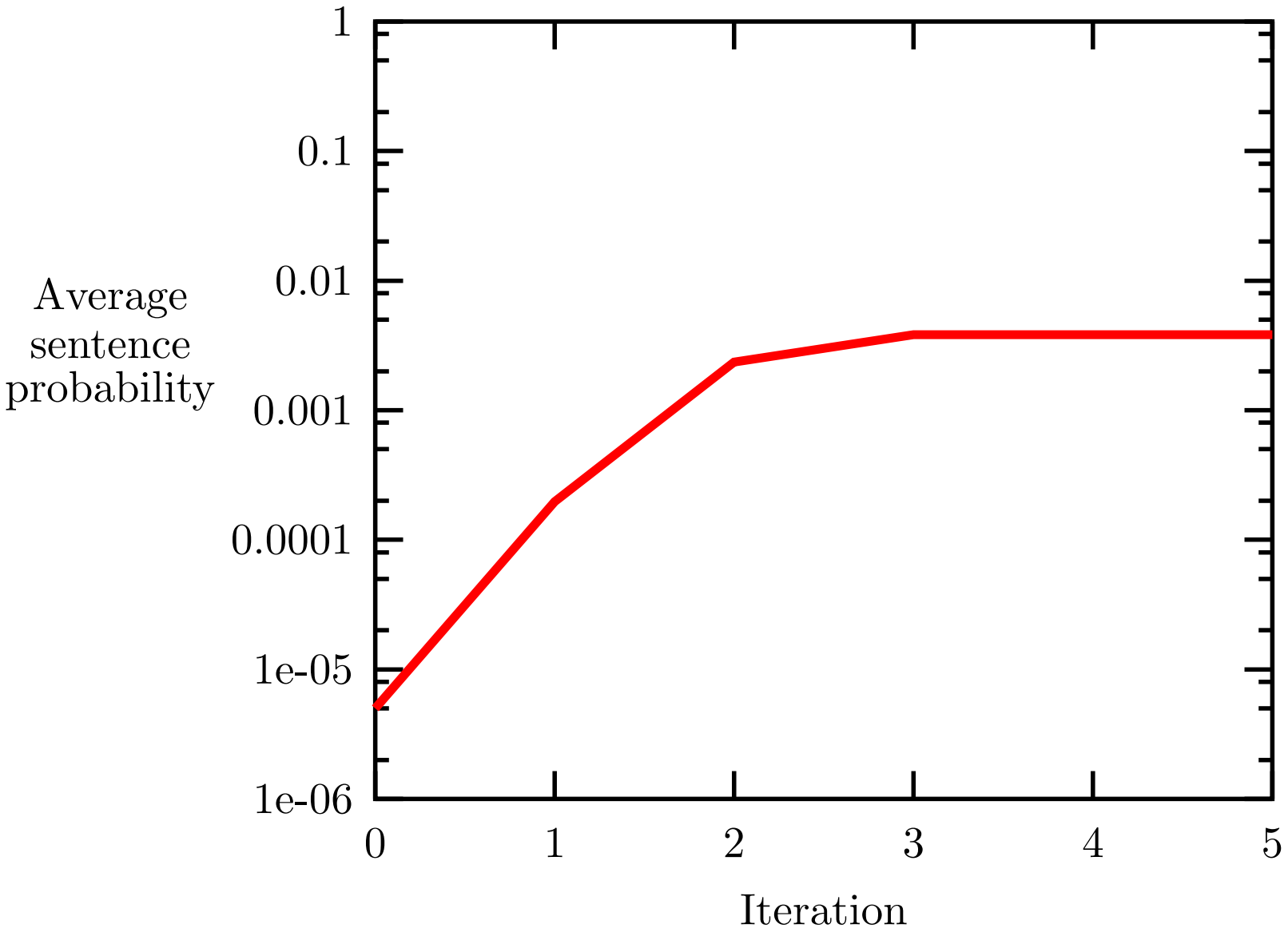
Probability of "English"



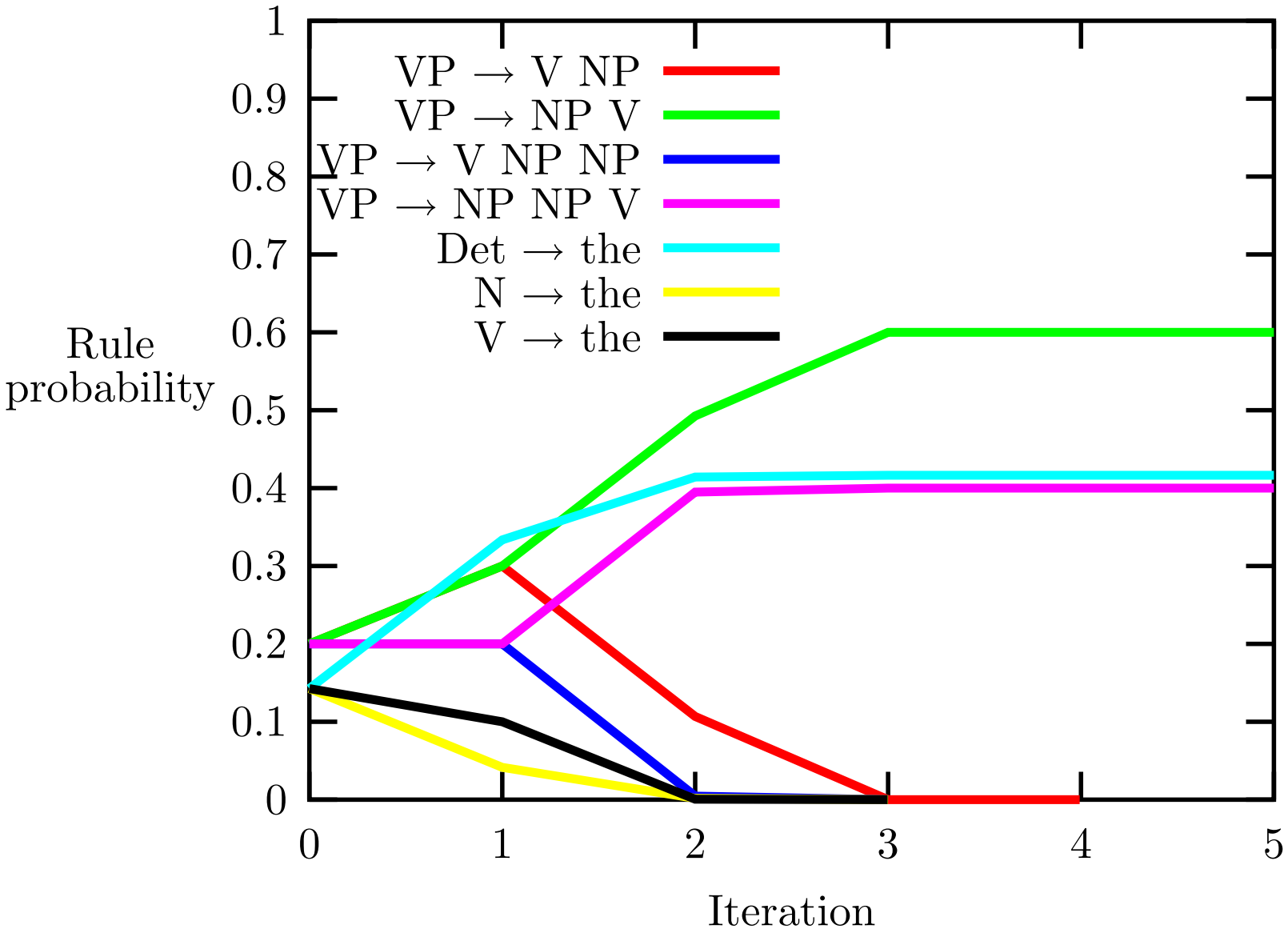
Rule probabilities from "English"



Probability of "Japanese"



Rule probabilities from "Japanese"

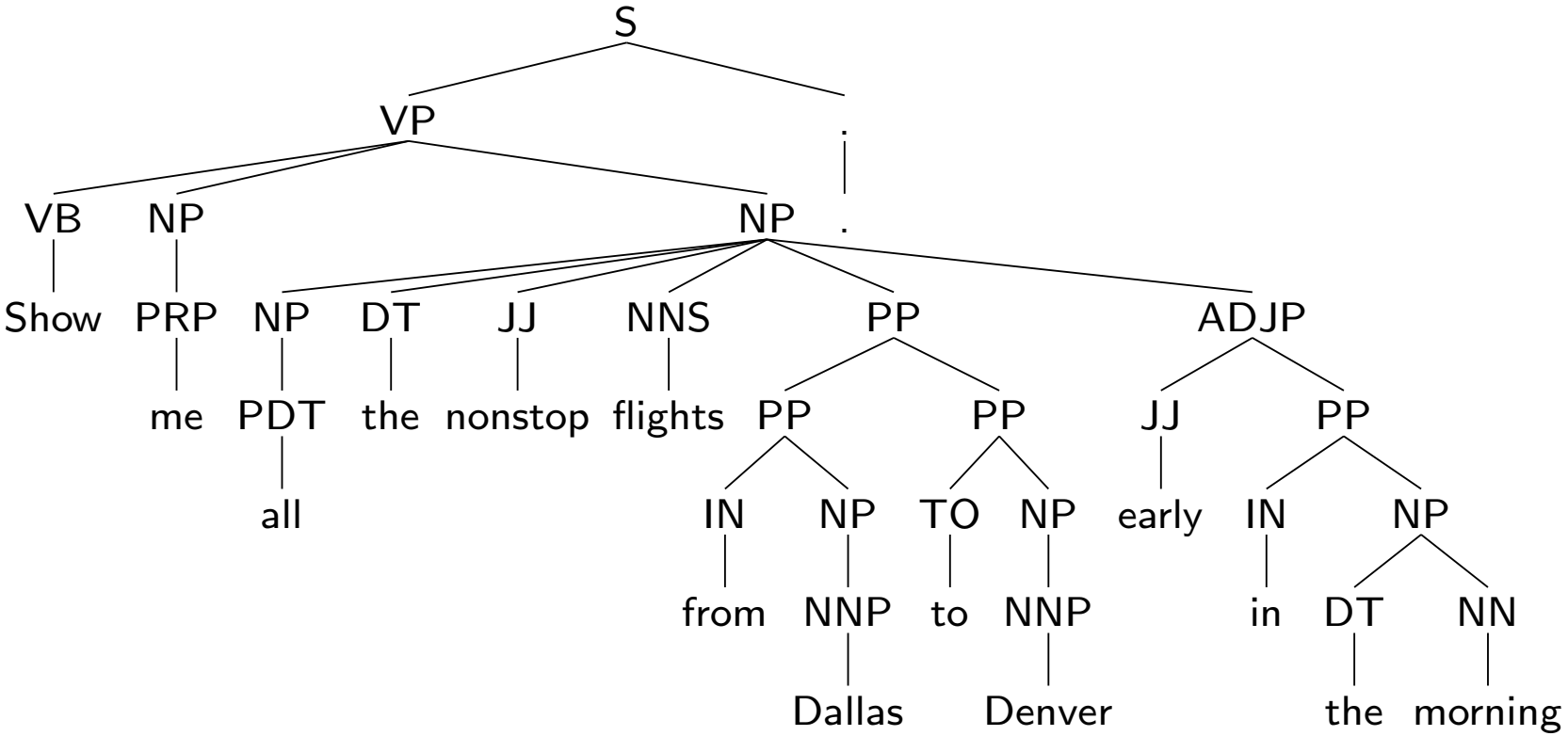


Learning in statistical paradigm

- 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

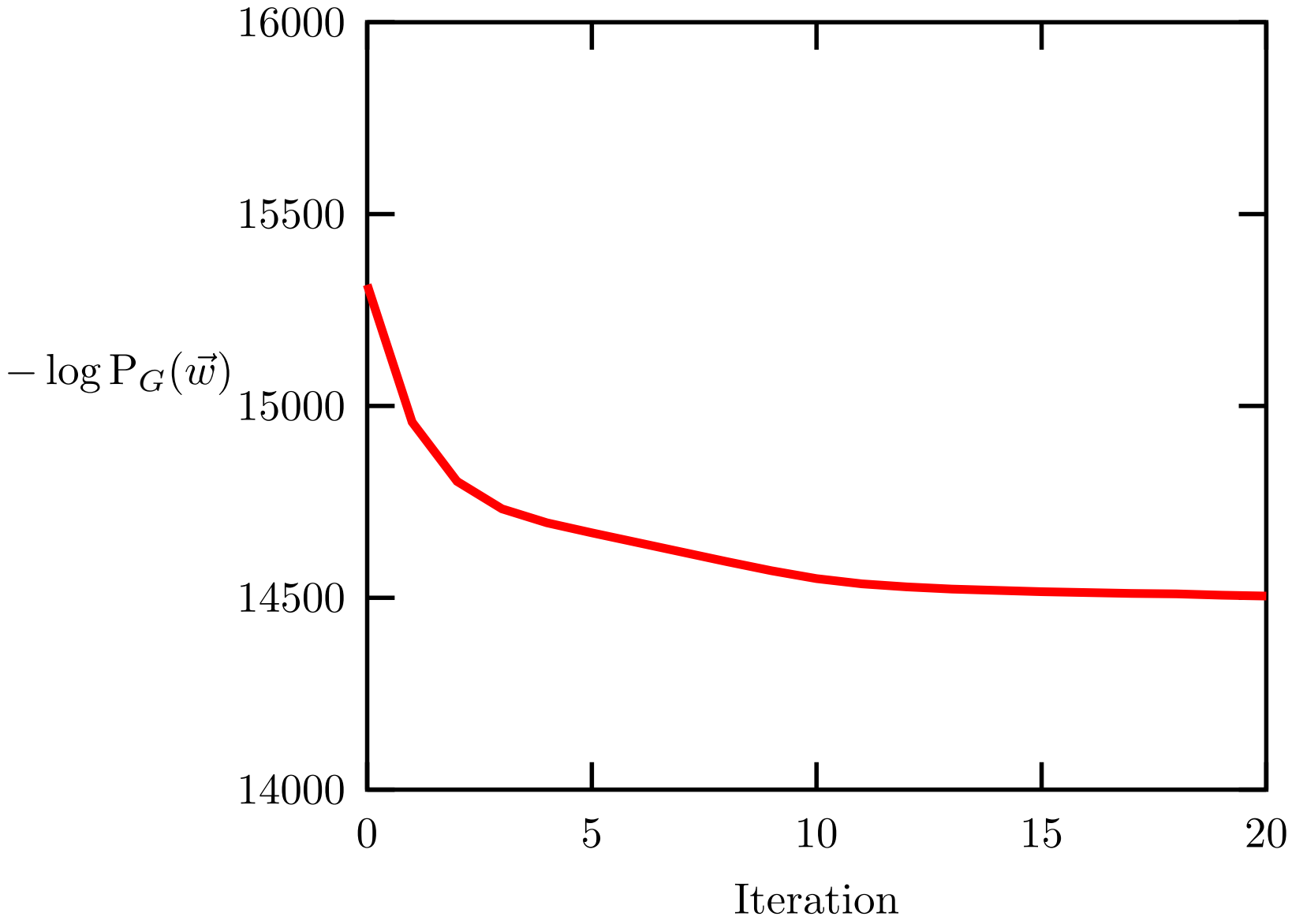
- ATIS treebank consists of 1,300 hand-constructed parse trees
- ignore the words (in this experiment)
- about 1,000 PCFG rules are needed to build these trees



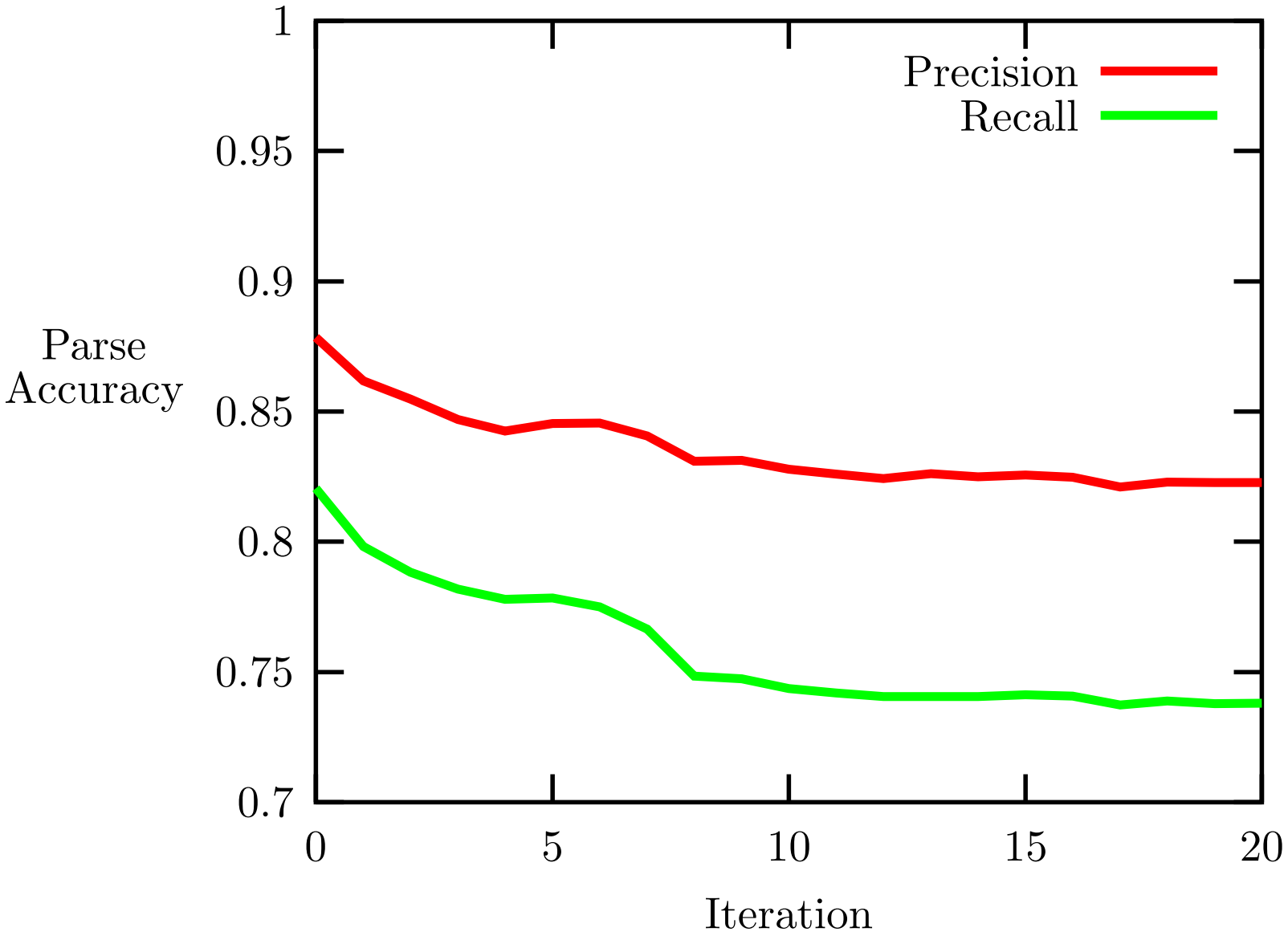
Experiments with EM

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.

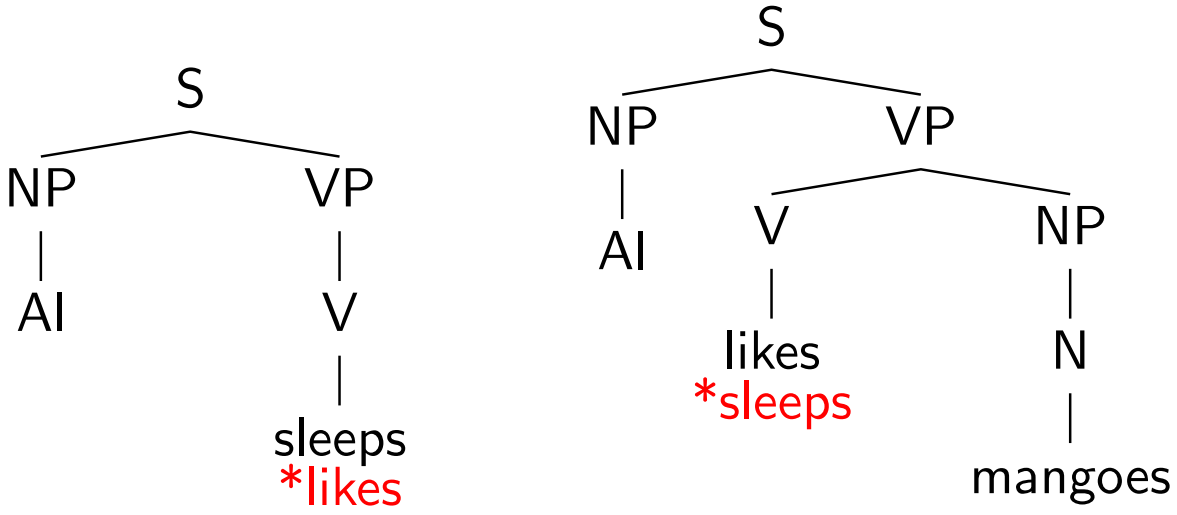
Talk outline

- What is computational linguistics?
- Probabilistic grammars
- Identifying phrase structure (parsing)
- Learning phrase structure
- More realistic grammars

Subcategorization

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

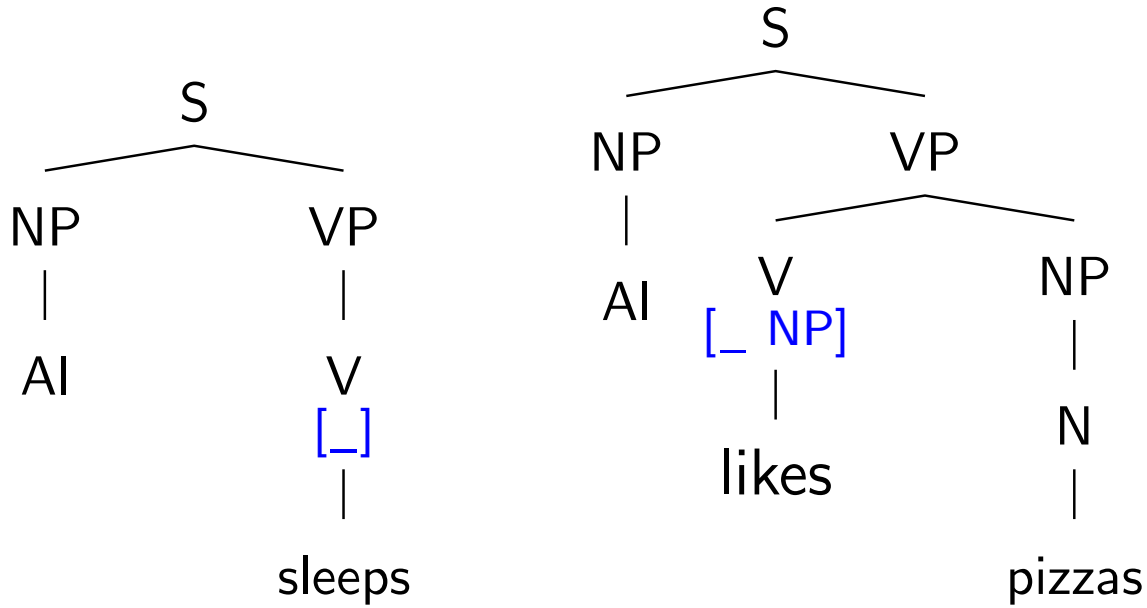
$R_3 = \{\mathsf{VP} \rightarrow \mathsf{V}, \mathsf{VP} \rightarrow \mathsf{V\,NP}, \mathsf{V} \rightarrow \mathsf{sleeps}, \mathsf{V} \rightarrow \mathsf{likes}, \ldots\}$



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.



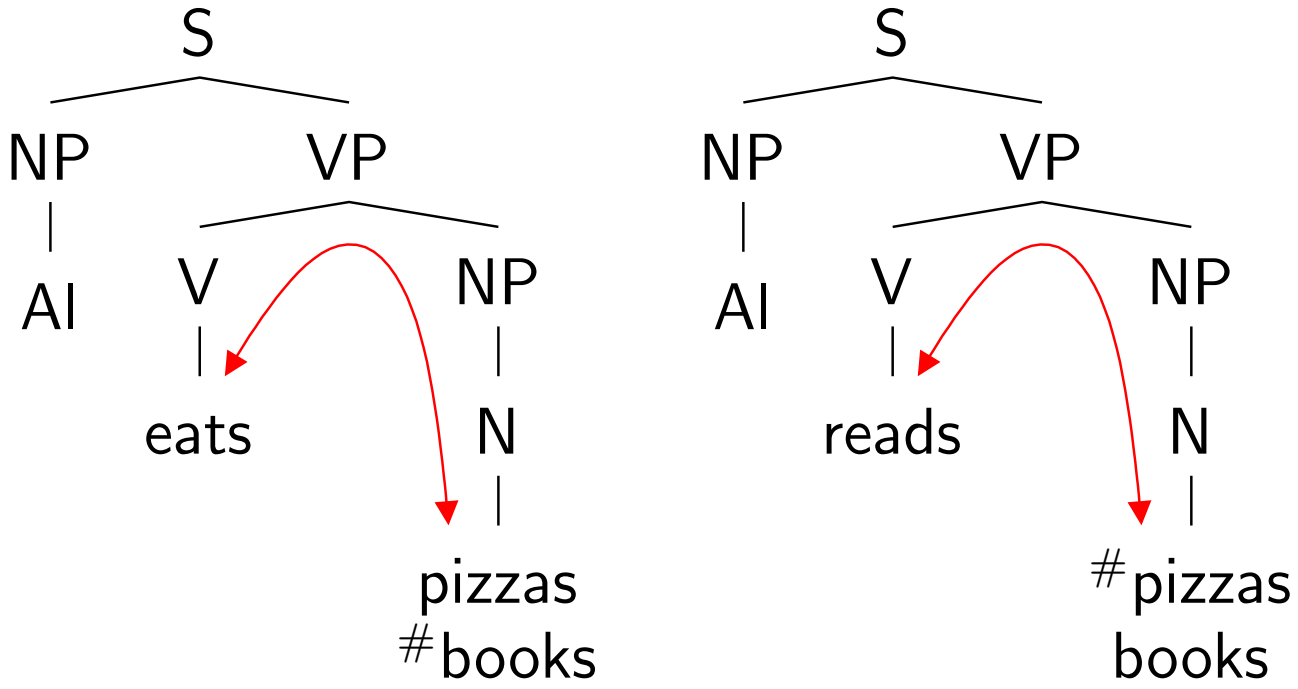
$R_4 = \{\mathsf{VP} \rightarrow \mathsf{V}_{[\_]}, \mathsf{VP} \rightarrow \mathsf{V}_{[\_\ \mathsf{NP}]}\ \mathsf{NP}, \mathsf{V}_{[\_]} \rightarrow \mathsf{sleeps}, \mathsf{V}_{[\_\ \mathsf{NP}]} \rightarrow \mathsf{likes}, \ldots\}$

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

Sparse data becomes a big problem; addressed with *regularization* and *smoothing*.

Selectional preferences

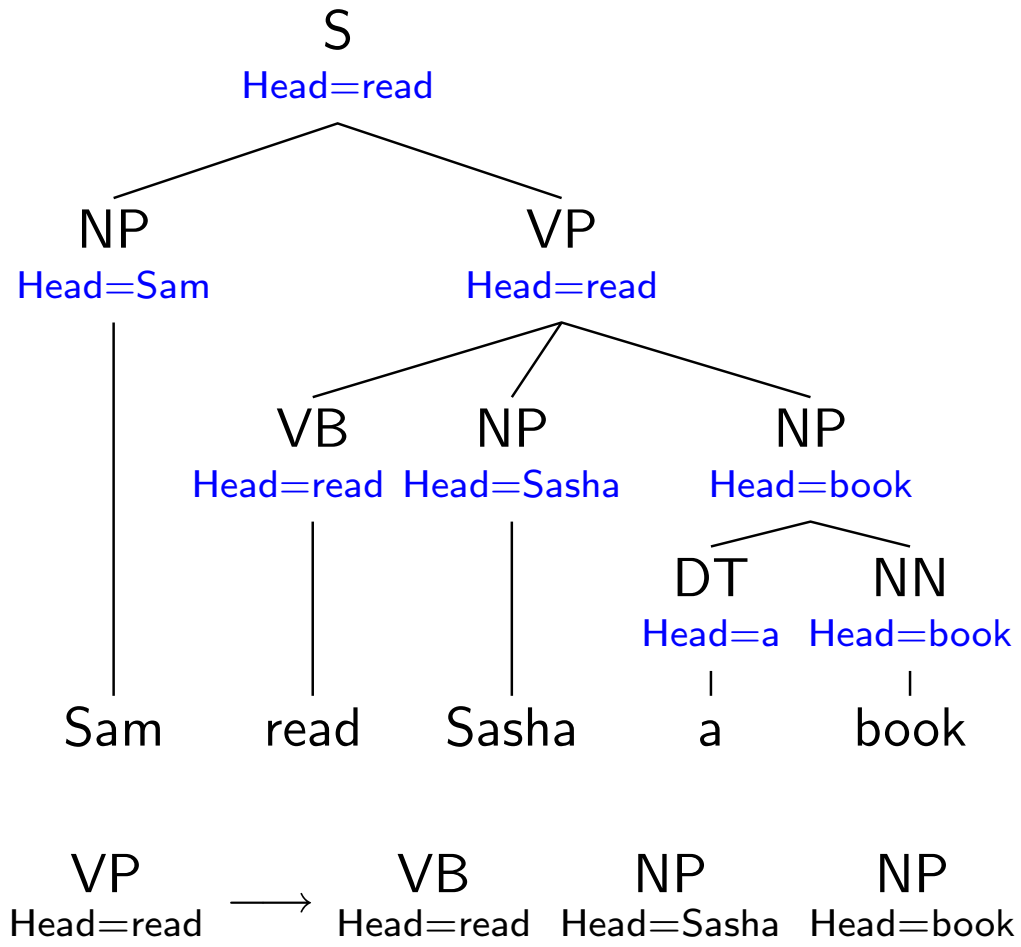
Head-to-head dependencies are an approximation to real-world knowledge.



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

AI drives a (#toy model) car

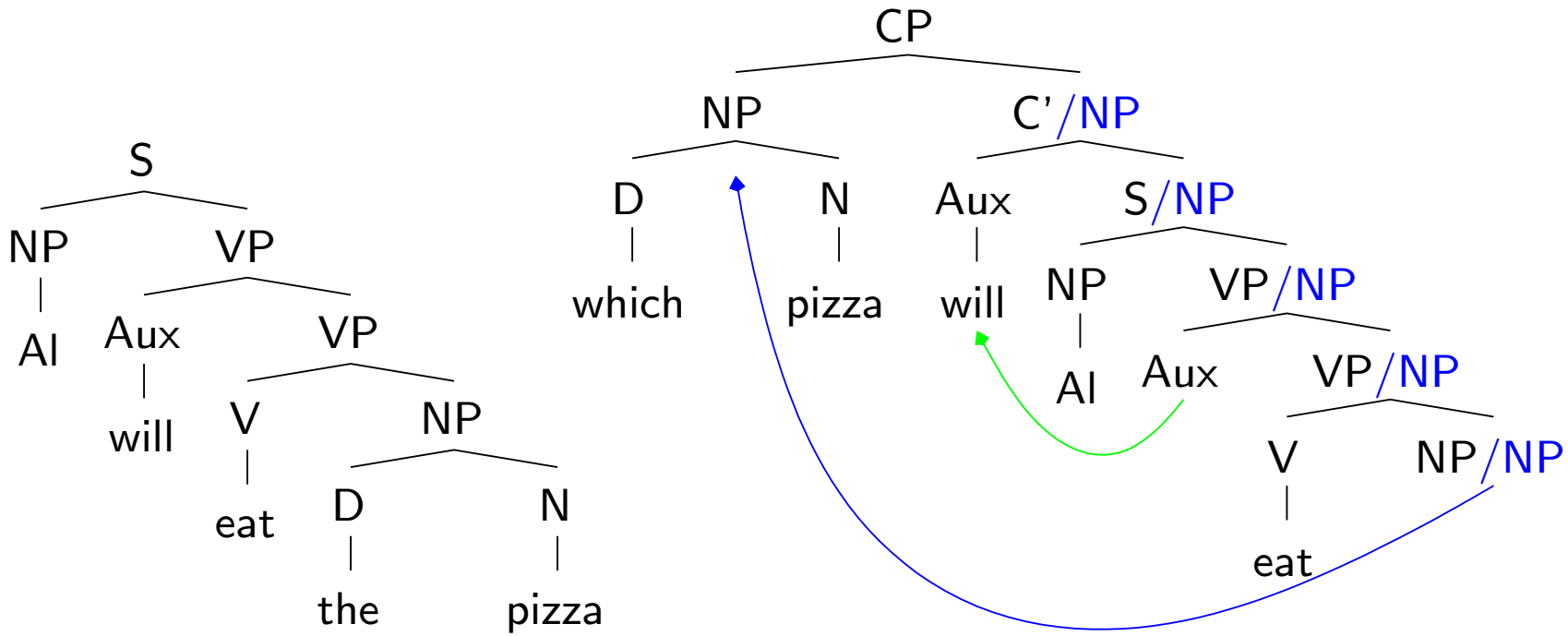
Head to head dependencies and bilexical rules



The number of possible rules grows rapidly; with a $\approx 10^4$ word vocabulary one might expect over 10^8 possible bilexical rules.

Sparse data is the big problem with grammars like this.

Nonlocal "movement" dependencies



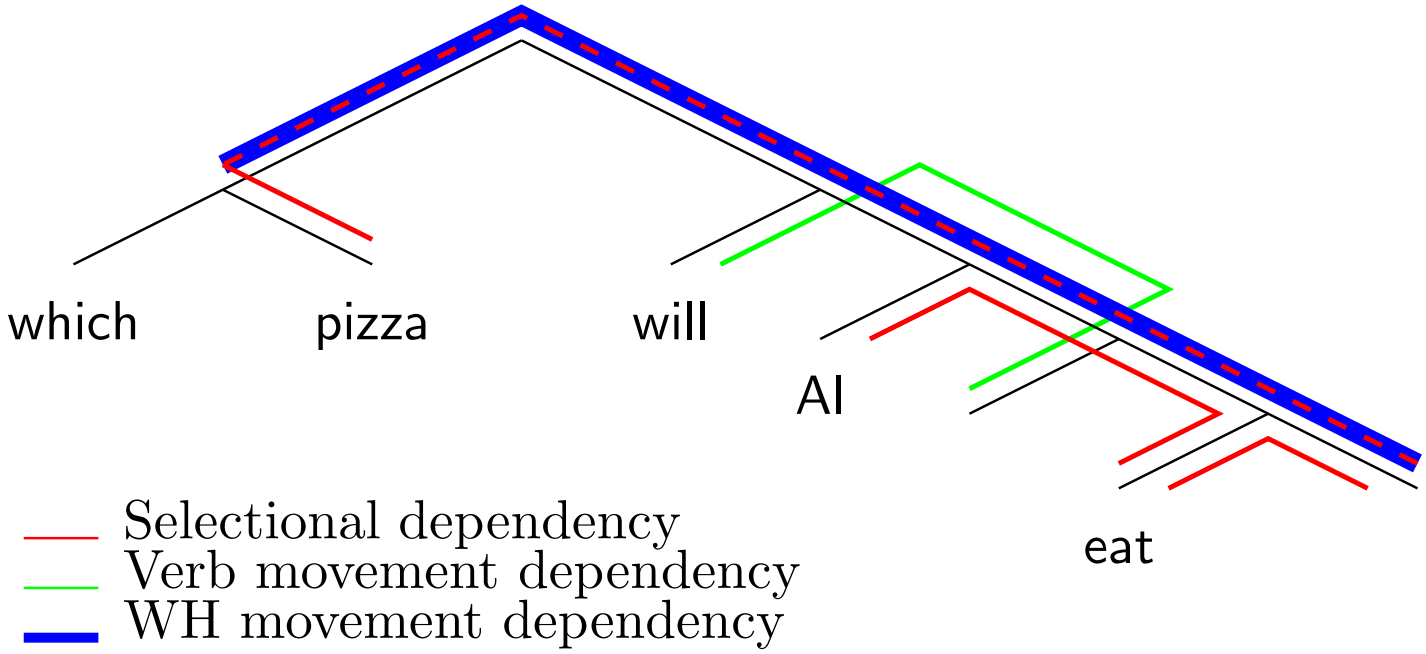
Subcategorization and selectional preferences are *preserved* under movement.

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

Nonterminal labels as communication channels

The nonterminal label is the *communication channel* between two parts of the tree.

Learn or design the structure of non-terminals so they pass the appropriate information around the tree.



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

Sparse data and therefore *smoothing* are major issues!

Summary and Conclusion

- Computational linguistics is great fun!
- . . . and maybe will help us understand deep things about language and the mind
- Language involves *rich compositional structure*
- . . . and *grammars* are a way of describing that structure
- *Probabilistic grammars* give us a systematic way of distinguishing more likely structures from less likely structures
- The number of parses (structures) can grow *exponentially* with sentence length
- . . . but there are *polynomial-time dynamic programming algorithms* for most of the important problems
- *Sparse data* is a big problem for learning realistic grammars

Talk outline

- What is computational linguistics?
- Probabilistic grammars
- Identifying phrase structure (parsing)
- Learning phrase structure
- More realistic grammars

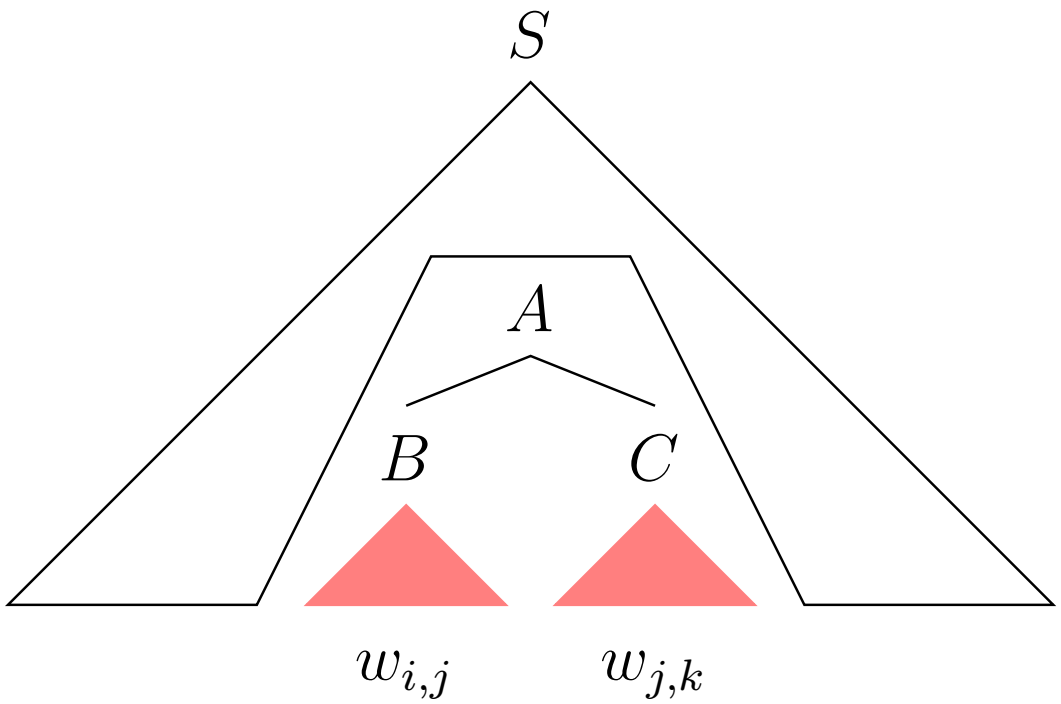
Applications of neural networks in CL

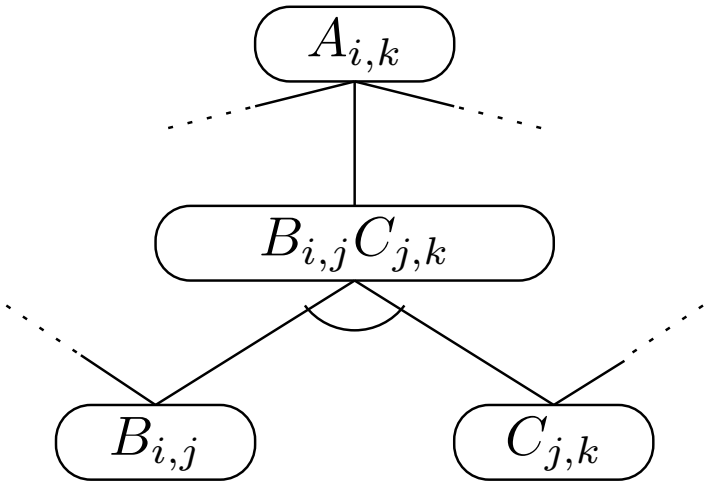
- The big problem: *how can NNs represent compositional structure?*
- NNs are sometimes used as *components* in larger models (e.g., predicting a node's label given the labels of its children)
  - other machine learning techniques often work better, and are computationally more efficient
- Applying *recurrent NN models of structure in time* has generally not been successful ("a dog walking on its hind legs")
  - Recurrent NNs seem to be equivalent to (factored) finite state machines

    $\Rightarrow$ incapable of capturing *two-dimensional nature* of phrase structure

"Brute force" NN simulation of CKY parsing

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





- Introduce a node for each $A_{i,k}$ and for each $(B_{i,j}, C_{j,k})$
- $(B_{i,j}, C_{j,k})$ computes $p(A \rightarrow B\,C)\mathrm{P}_G(B \Rightarrow^\star w_{i,j})\mathrm{P}_G(C \Rightarrow^\star w_{j,k})$
- $A_{i,k}$ sums over all $A \rightarrow B\,C$ and $(B_{i,j}, C_{j,k})$
- Asynchronous (non-real-time) model with unnatural parameter tying

Incremental parsing algorithms

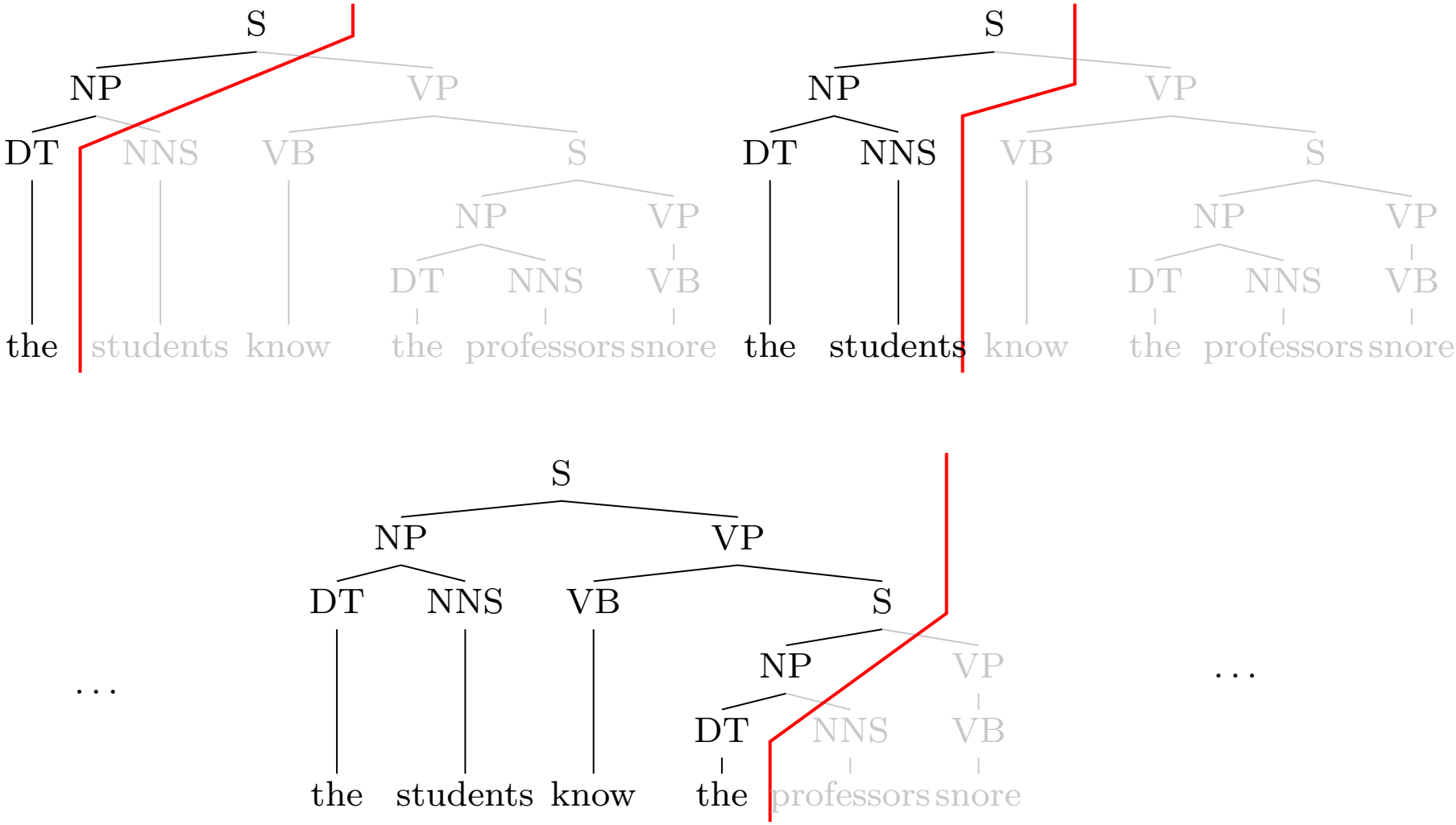
- Parsing involves incrementally constructing the parse tree
- We require that words are incorporated in left-to-right order
- Because of the two-dimensional nature of phrase structure trees, there are many ways of parsing incrementally

  Top-down: Posit parent *before* any children

  Bottom-up: Posit parent *after* all children

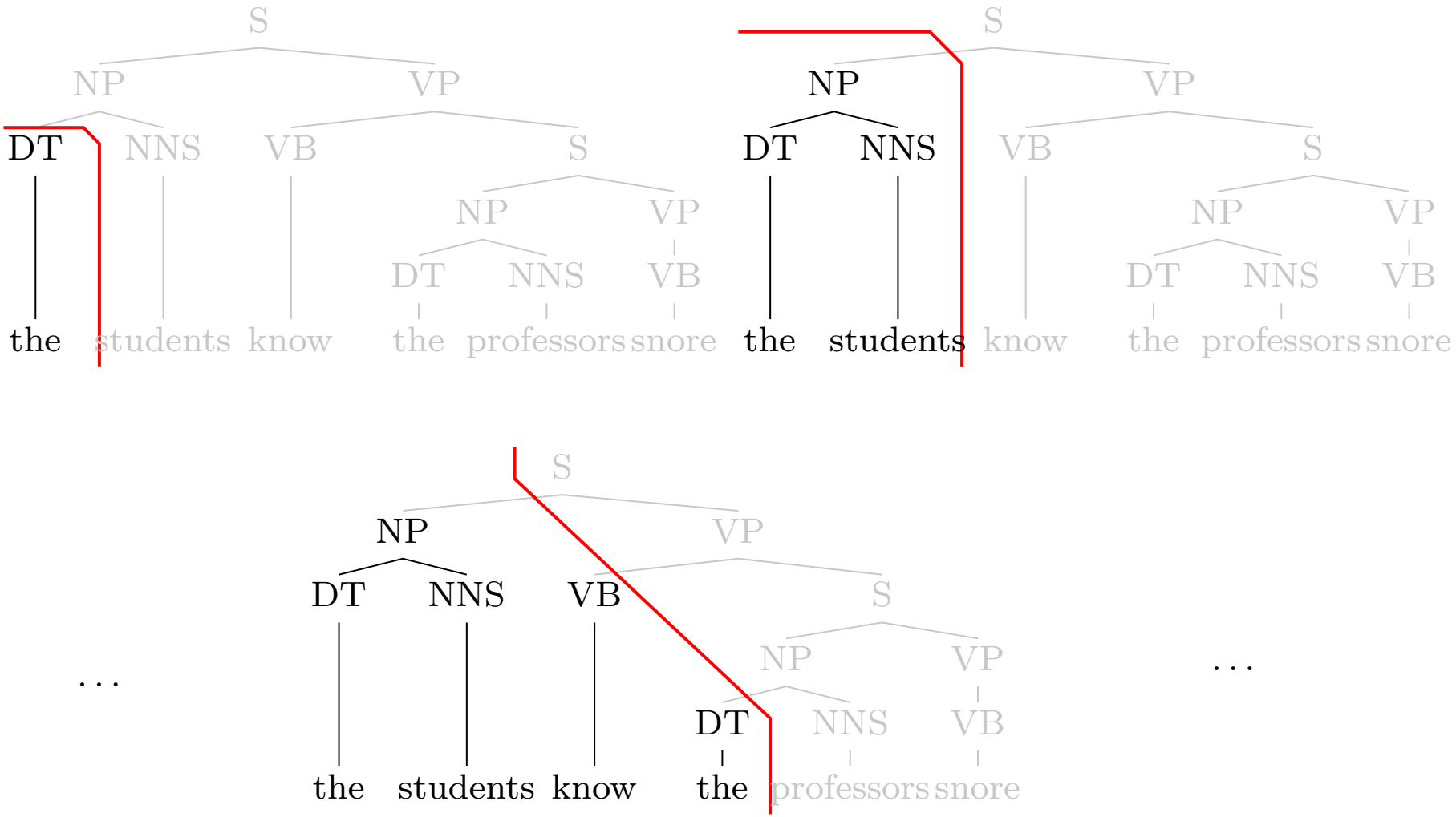
  Left-corner: Posit parent *after first* child
- Every incremental parsing algorithm corresponds to different sets of *cuts through edges in the phrase structure tree*
- Only *the edges on such cuts are active*; the rest of the tree can be ignored (as far as parsing goes)

Top-down parsing

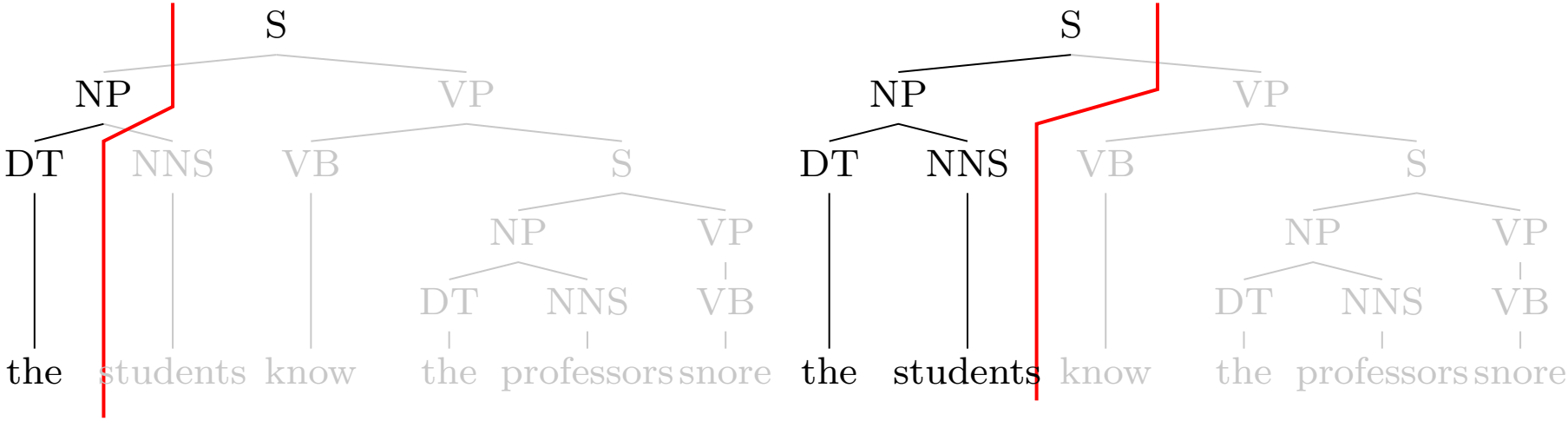


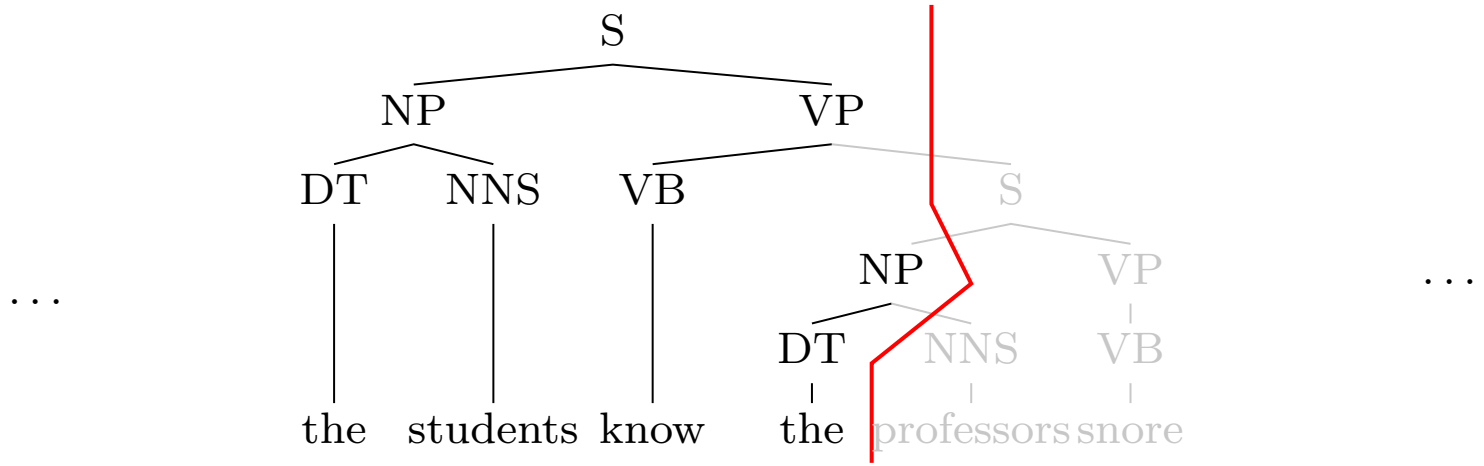
Top-down parsing is the only parsing algorithm that always maintains a *connected partial parse tree*

Bottom-up parsing



Left-corner parsing





Incremental parsing and NNs

+ Only the small fraction of nodes in the parse tree lying on the cut are *active* at any point in the string
  - Can these be represented and updated by a recurrent NN? (Charniak)
- These nodes are manipulated with a LIFO stack discipline
  - Unnatural in a NN architecture