An introduction to grammars and parsing

Mark Johnson

Microsoft Research / Brown University

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

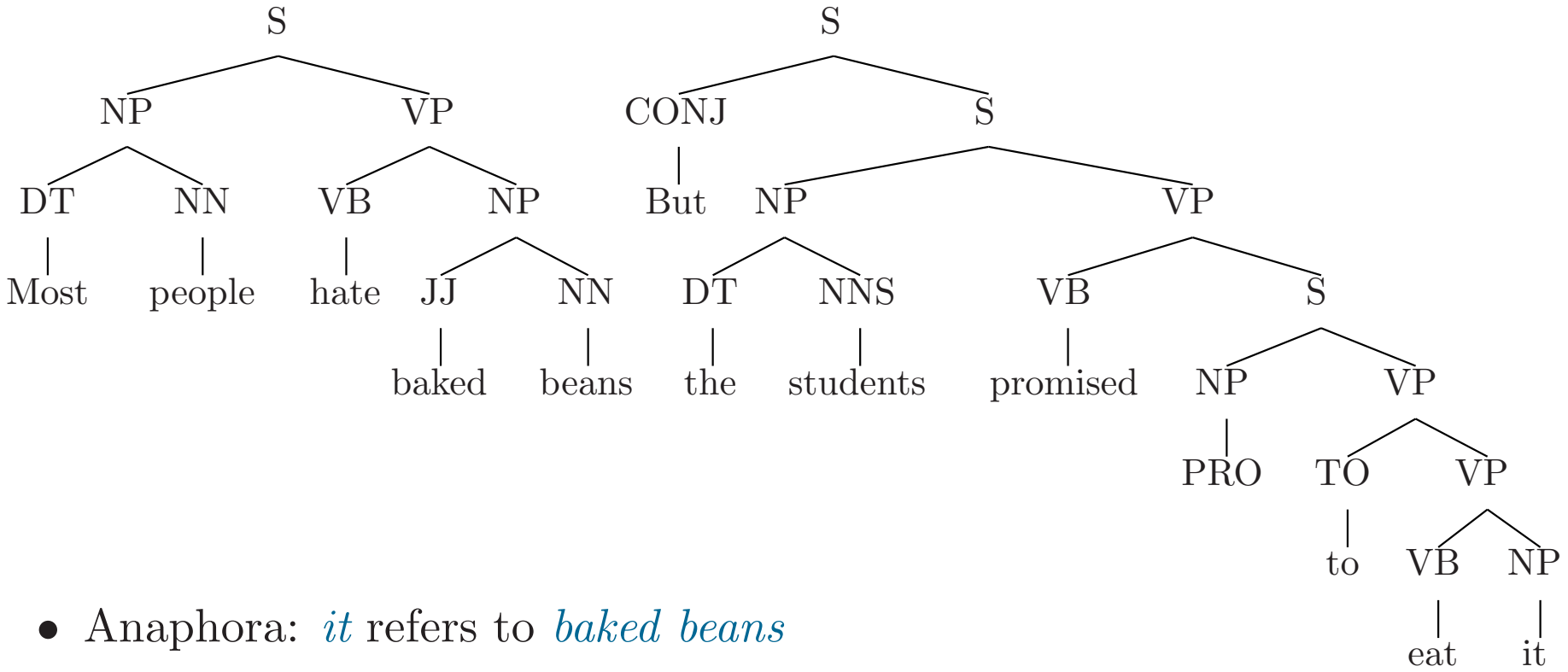
– We know little about neural networks that might support linguistic computations

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

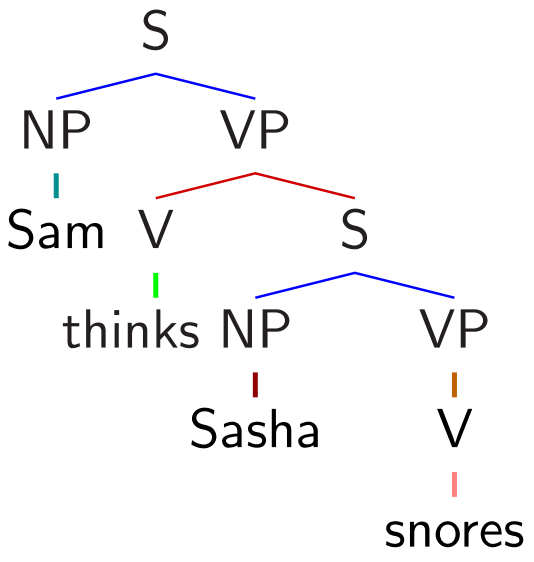
A *context-free grammar* $G = (\mathcal{V}, \mathcal{S}, s, \mathcal{R})$ consists of:

- $\mathcal{V}$, 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 $A \rightarrow X_1 \ldots X_n \in \mathcal{R}$

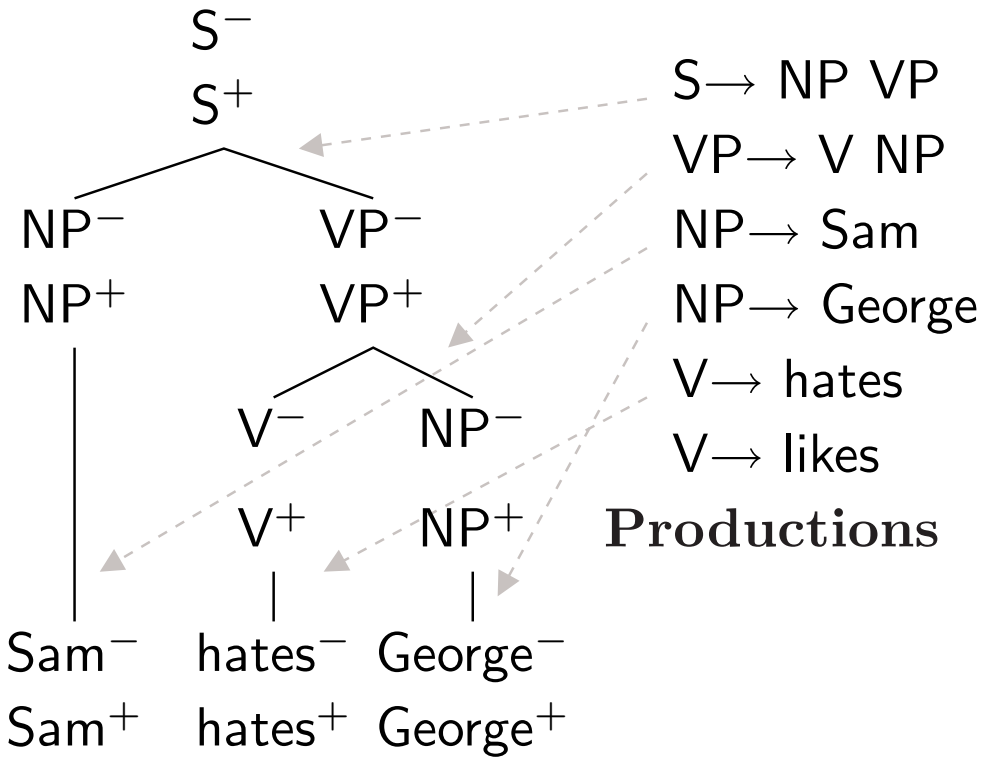
G generates a string $w \in \mathcal{V}^\star$ iff w is the *terminal yield* of a tree generated by G



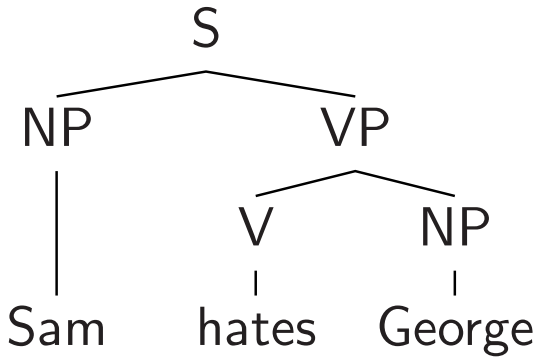
Productions

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

CFGs as "plugging" systems



"Pluggings"

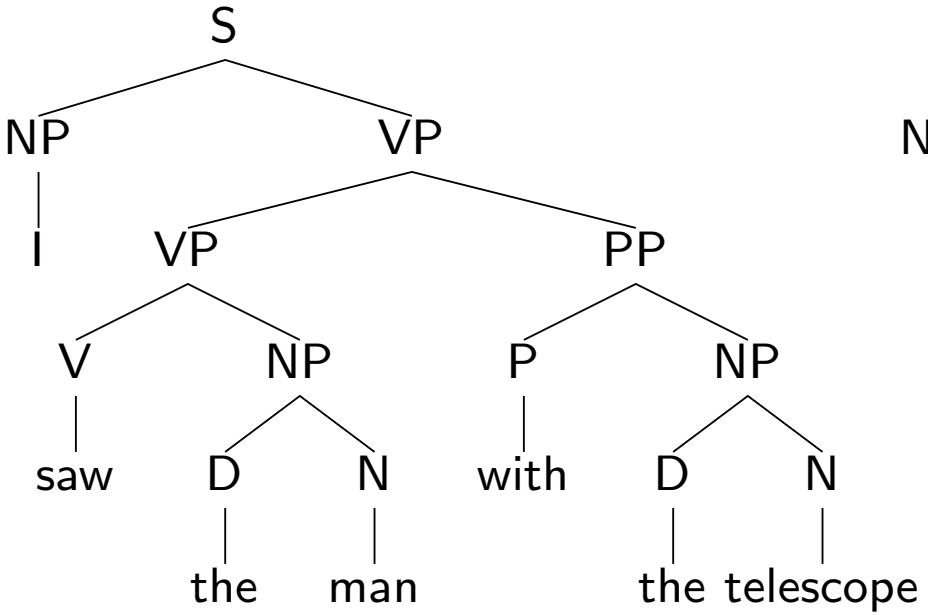


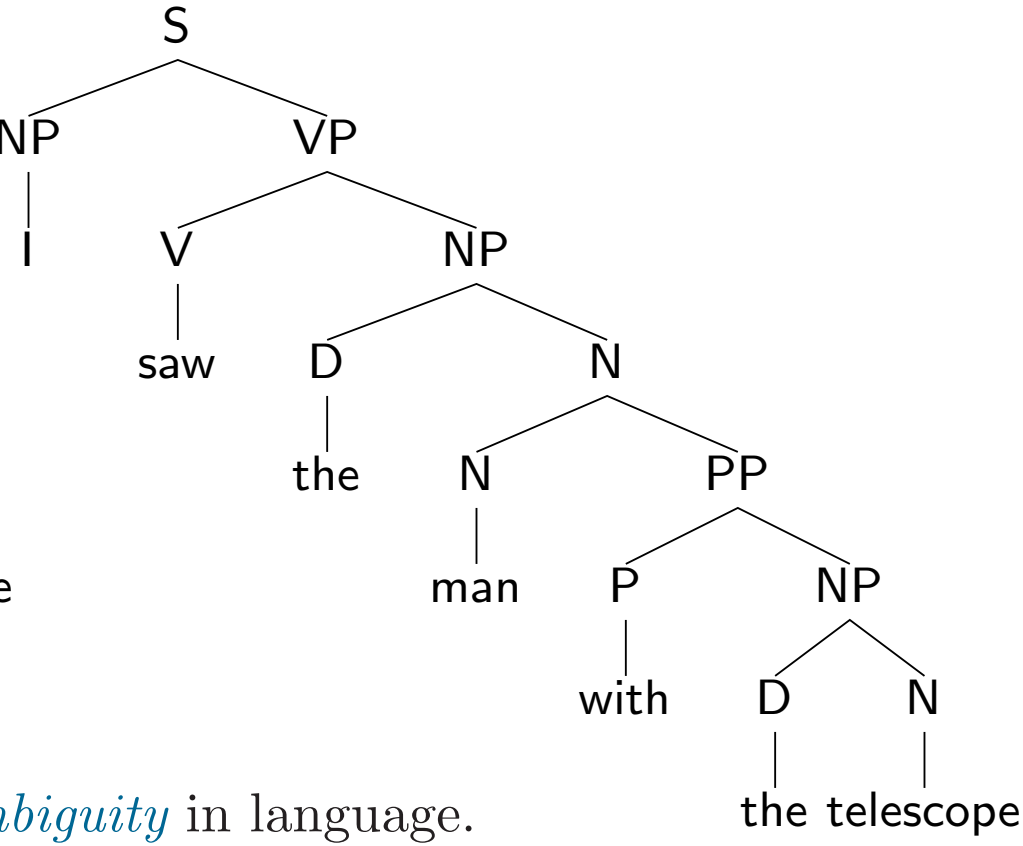
Resulting tree

- 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 $A \to \beta \in \mathcal{R}$ 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 $\Rightarrow_G$.

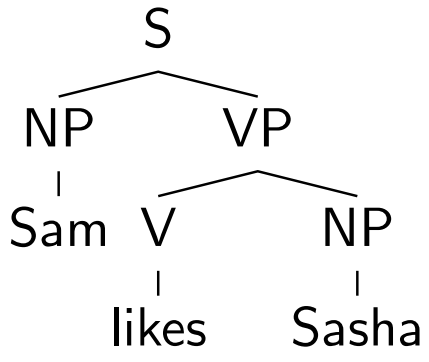
The *language generated* by G is $\{w : s \Rightarrow^\star w, w \in \mathcal{V}^\star\}$.

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

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 *normal forms*



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

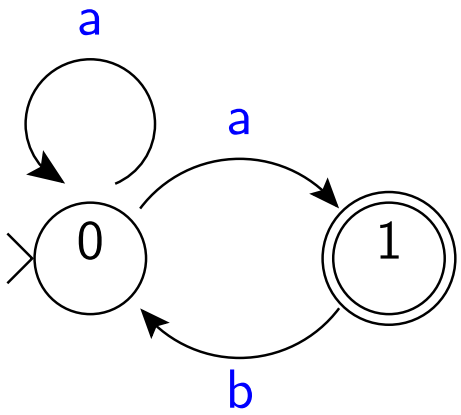
1.0	S $\rightarrow$ NP VP	1.0	VP $\rightarrow$ V
0.75	NP $\rightarrow$ George	0.25	NP $\rightarrow$ Al
0.6	V $\rightarrow$ barks	0.4	V $\rightarrow$ snores

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

(Mealy) finite-state automata

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.

What's interesting about finite-state automata?

- Finite state automata are probably the simplest devices that generate an infinite number of strings
- They are conceptually and computationally simpler than context-free grammars
- They are expressive enough for many useful tasks:
  - Speech recognition
  - Phonology and morphology
  - Lexical processing
- Very large FSA can be built and used very efficiently
- Good software tools exist for using and manipulating FSA

Probabilistic Mealy automata

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

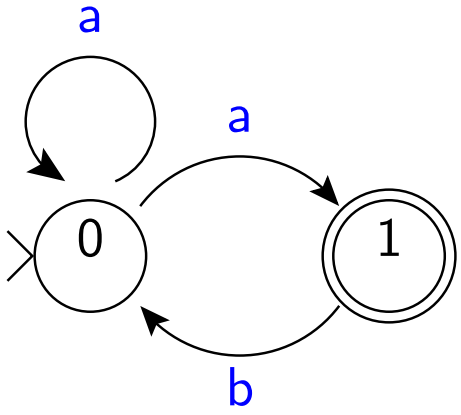
- terminals $\mathcal{V}$, states $\mathcal{S}$ and start state $s_0 \in \mathcal{S}$ as before,
- $p_f(s)$, the probability of *halting at state* $s \in \mathcal{S}$, and
- $p_m(v, s'|s)$, the probability of *moving from* $s \in \mathcal{S}$ *to* $s' \in \mathcal{S}$ *and emitting a* $v \in \mathcal{V}$.

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

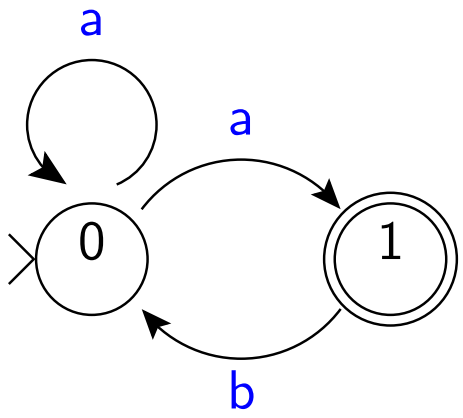


Probabilistic FSA as PCFGs

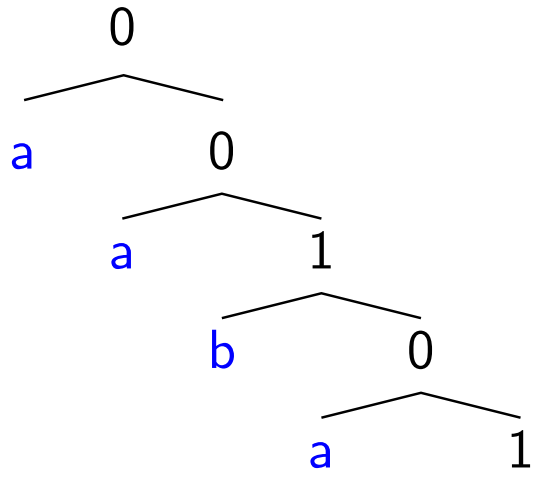
Given a Mealy PFSA $M = (\mathcal{V}, \mathcal{S}, s_0, p_f, p_m)$, let G_M have the same terminals, states and start state as M, and have productions

- $s \to \epsilon$ 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 $v \in \mathcal{V}$

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



Mealy FSA



PCFG parse of aaba

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

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

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

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

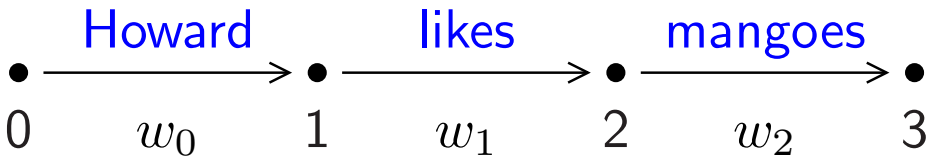
String positions

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 $0 \leq i \leq j \leq n$.

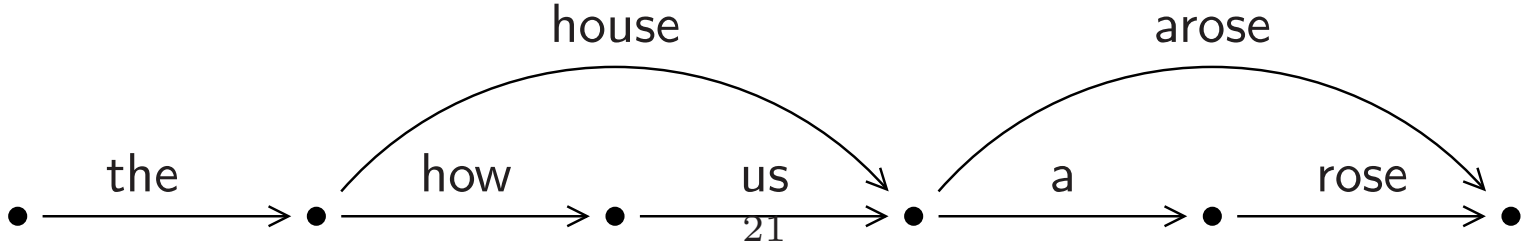
$w_{i,j}$ represents the substring $w_i \dots w_{j-1}$



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 $i = 0, \ldots, n-1$

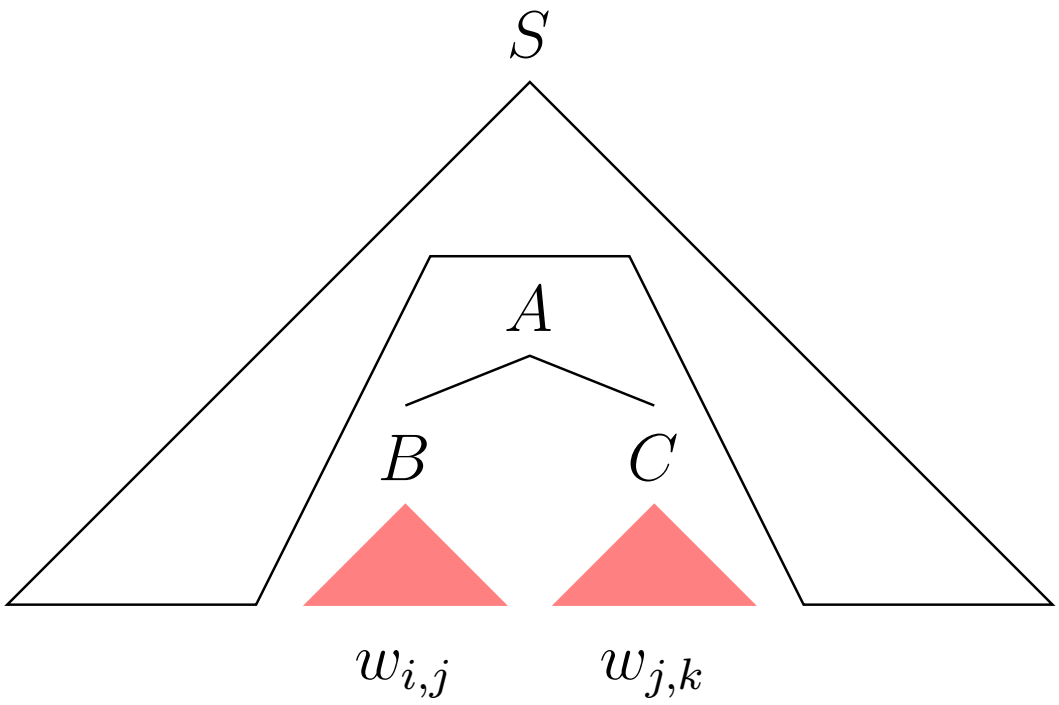
Recursion: $\mathrm{P}(A \Rightarrow^\star w_{i,k})$

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

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

Dynamic programming recursion

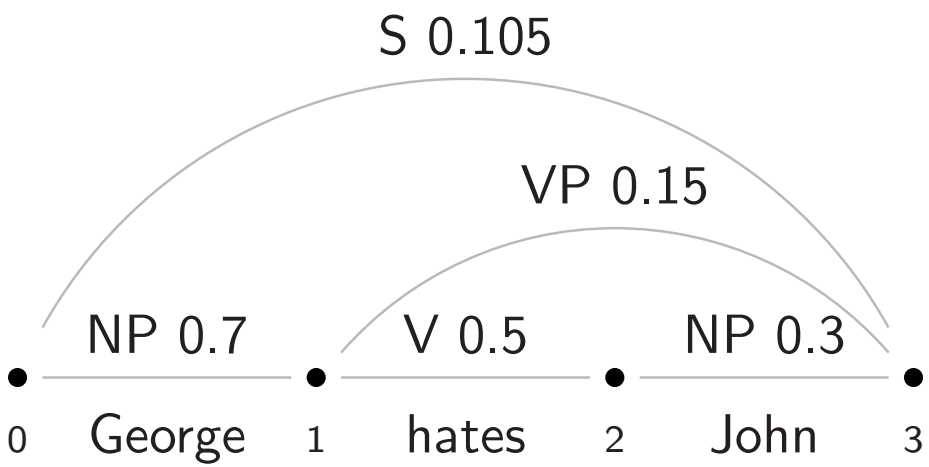
$$\mathrm{P}_G(A \Rightarrow^\star w_{i,k}) = \sum_{j=i+1}^{k-1} \sum_{A \rightarrow B\,C \in \mathcal{R}(A)} p(A \rightarrow B\,C)\mathrm{P}_G(B \Rightarrow^\star w_{i,j})\mathrm{P}_G(C \Rightarrow^\star w_{j,k})$$



$\mathrm{P}_G(A \Rightarrow^\star w_{i,k})$ 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 0.105
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

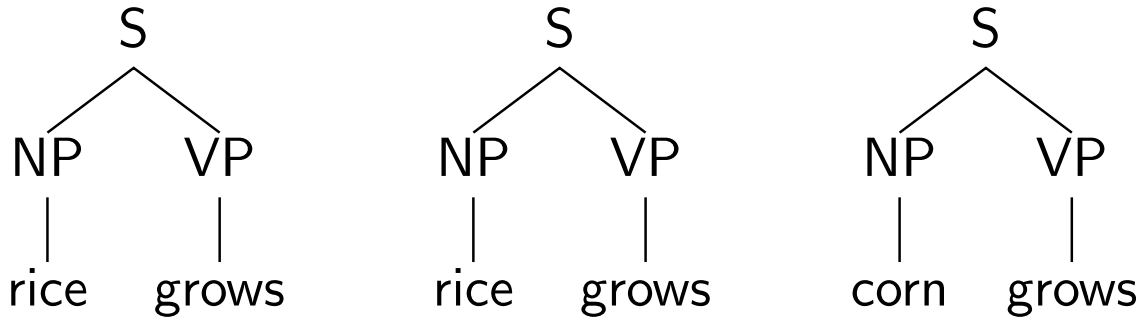
"Empiricist": Collect a corpus of examples, linguists annotate them with relevant information, a machine learning algorithm extracts generalizations

- 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 θ from the *visible data* x is difficult, but estimating θ from augmented data $z = (x, y)$ is easier (y is the *hidden data*).

The EM algorithm given visible data x:

1. guess initial value θ_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" $\mathrm{P}_{\theta_i}(y|x)$

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

The EM algorithm finds the MLE $\hat{\theta} = \mathrm{argmax}_\theta \mathrm{P}_\theta(x)$ of the *visible data* x.

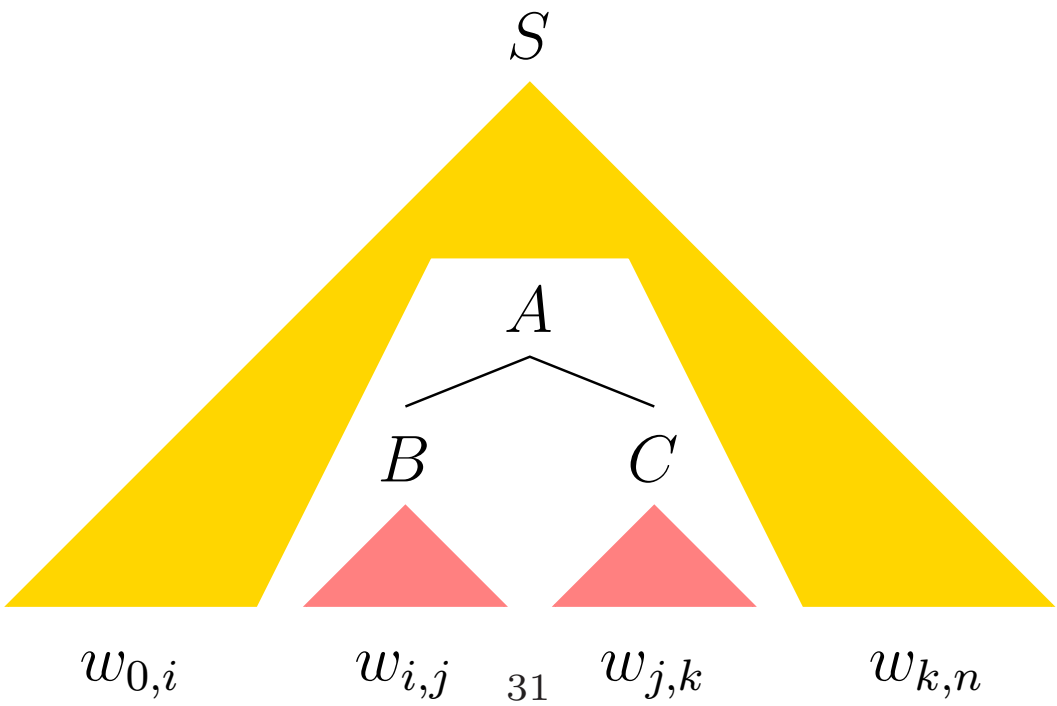
Sometimes it is not necessary to explicitly generate the pseudo-data (x, y); often it is possible to perform the maximization step directly from *sufficient statistics* (for PCFGs, the *expected production frequencies*)

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

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

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

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



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

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

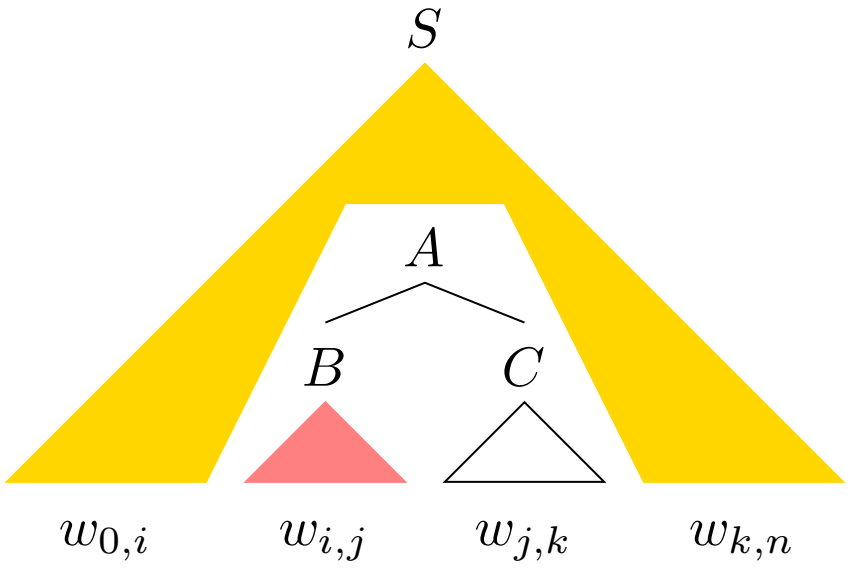
$$+ \sum_{l=k+1}^{n} \sum_{\substack{A,D \in \mathcal{S} \\ A \to C\, D \in \mathcal{R}}} \mathrm{P}(S \Rightarrow^\star w_{0,j}\, A\, w_{l,n}) p(A \to C\, D) \mathrm{P}(D \Rightarrow^\star w_{k,l})$$

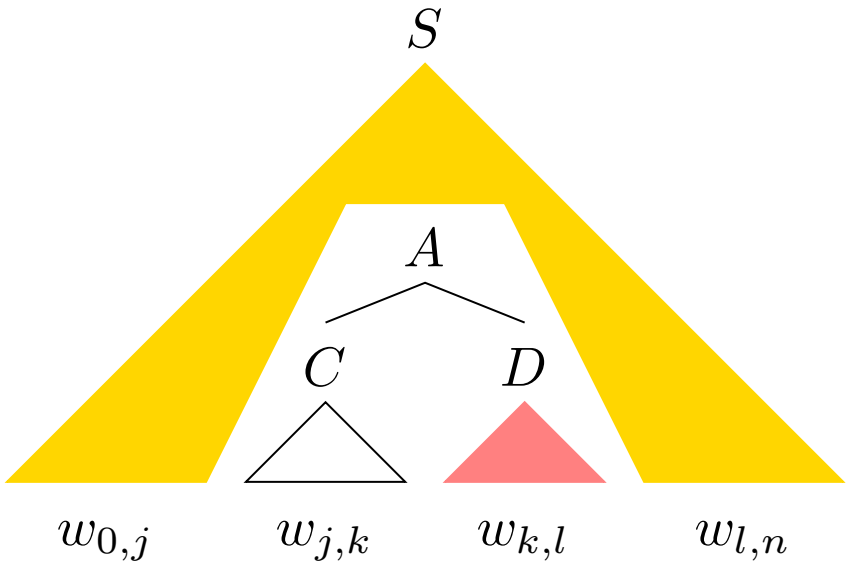
Recursion in $\mathrm{P}_G(S \Rightarrow^\star w_{0,i}\, A\, w_{k,n})$

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

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

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





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

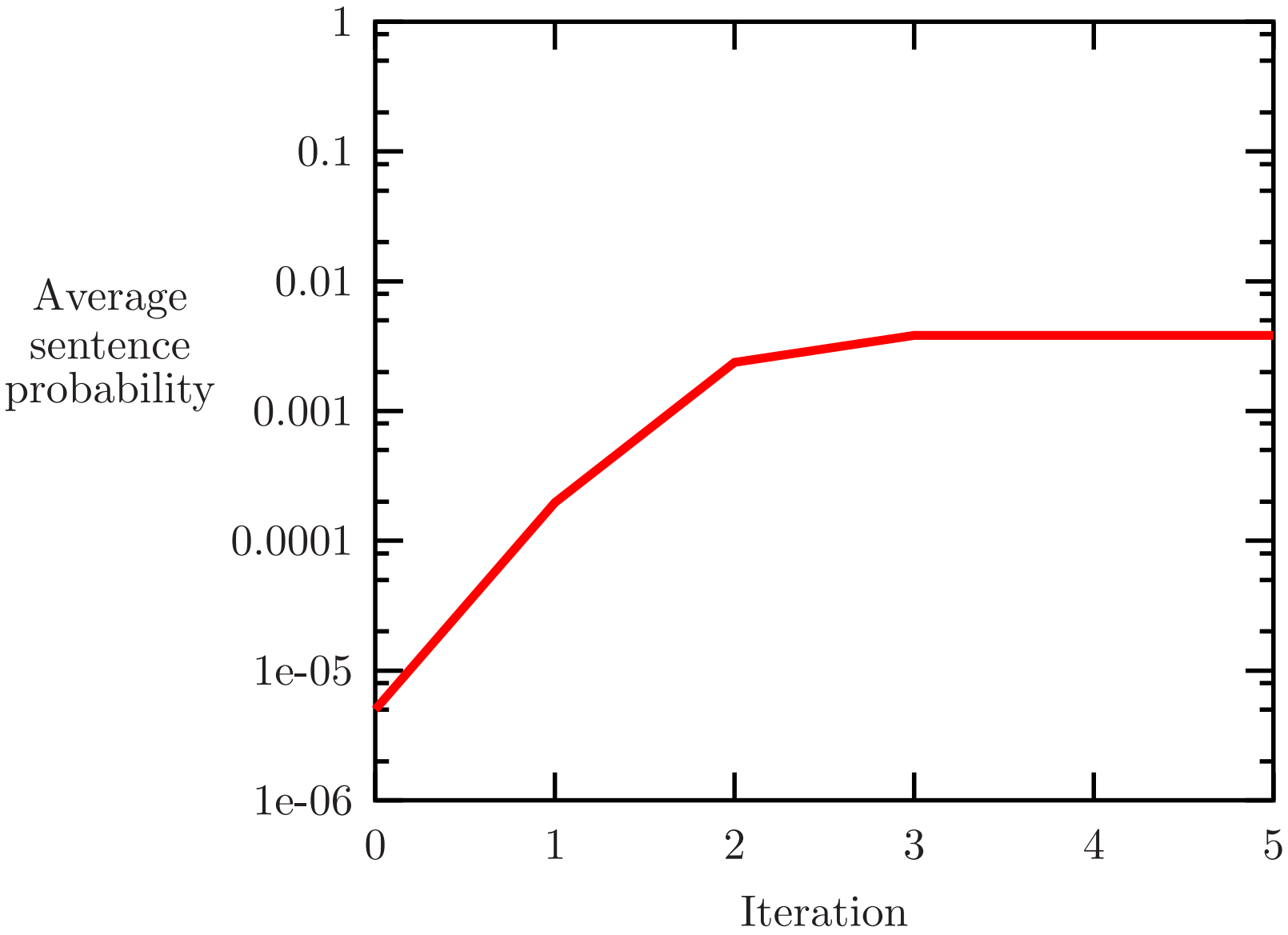
"English" input

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

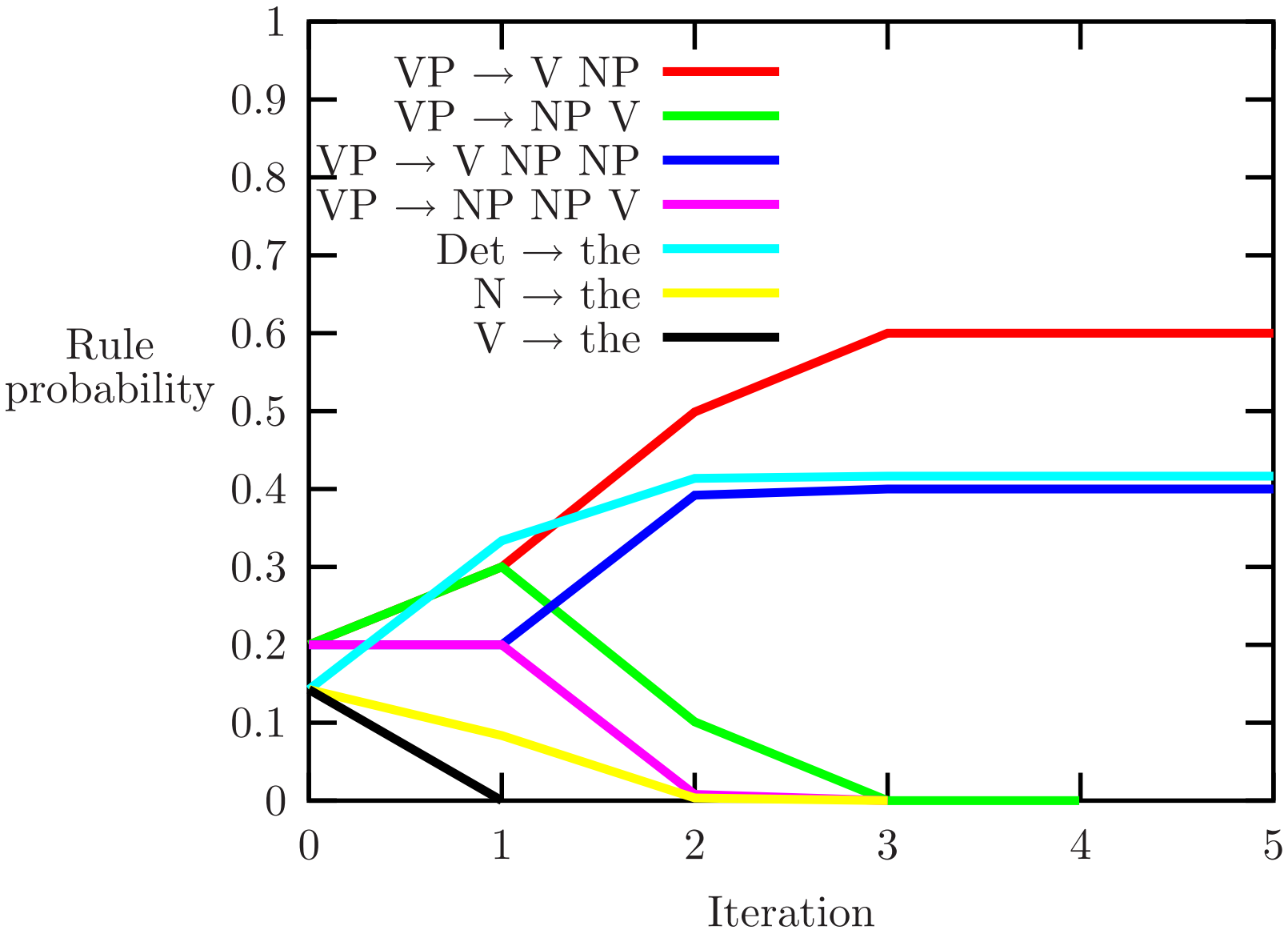
"pseudo-Japanese" input

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

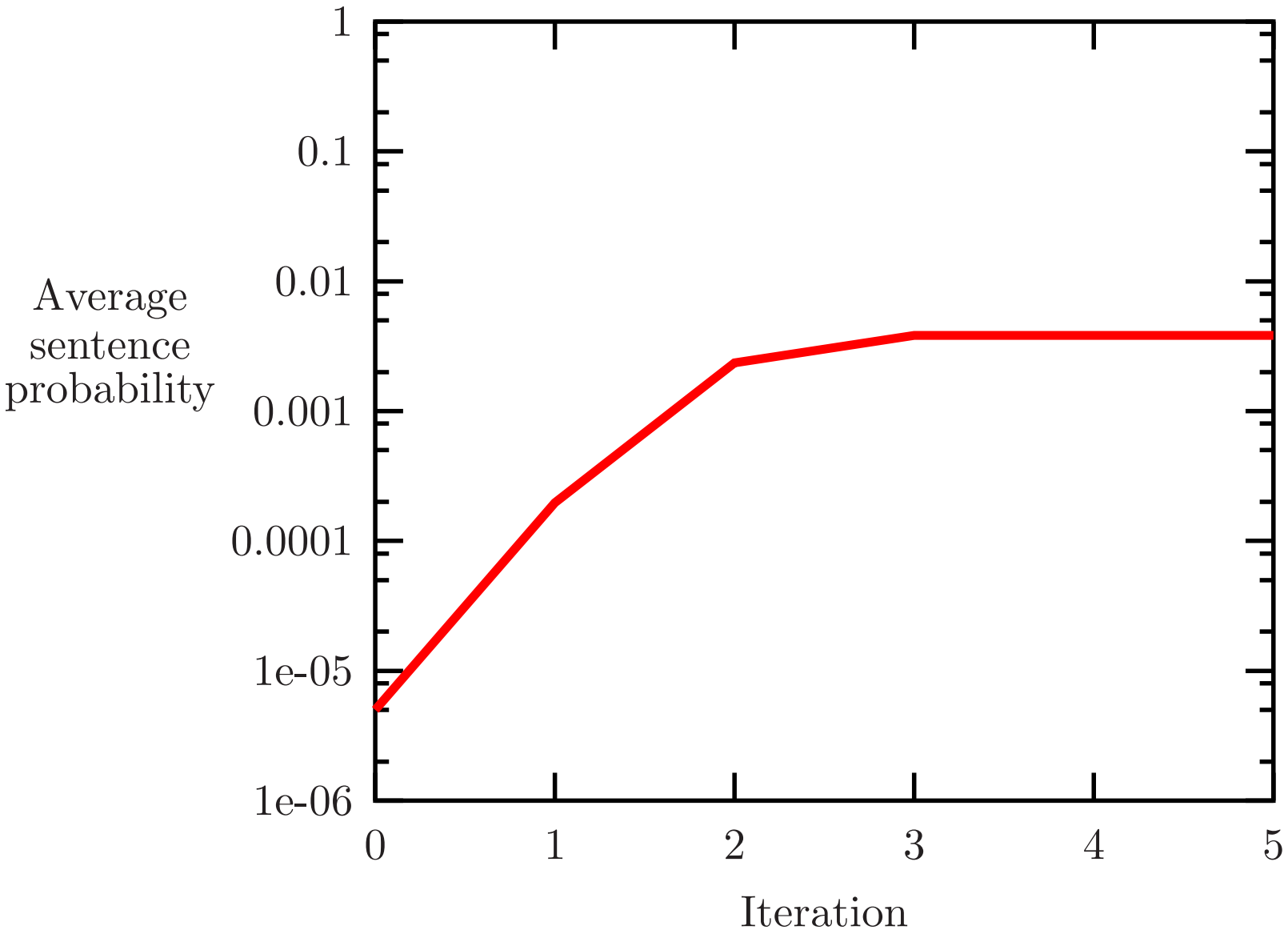
Probability of "English"



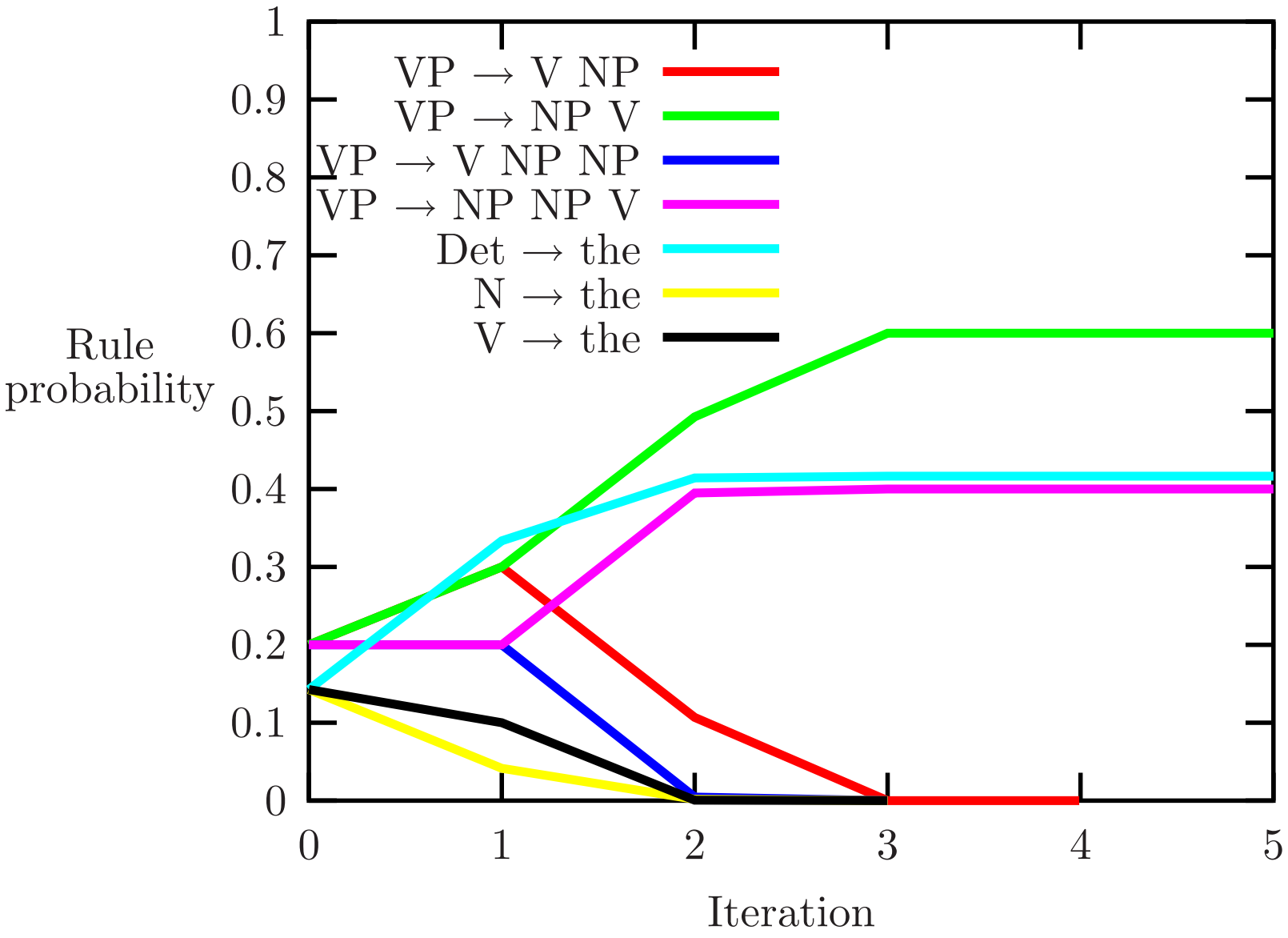
Rule probabilities from "English"



Probability of "Japanese"



Rule probabilities from "Japanese"

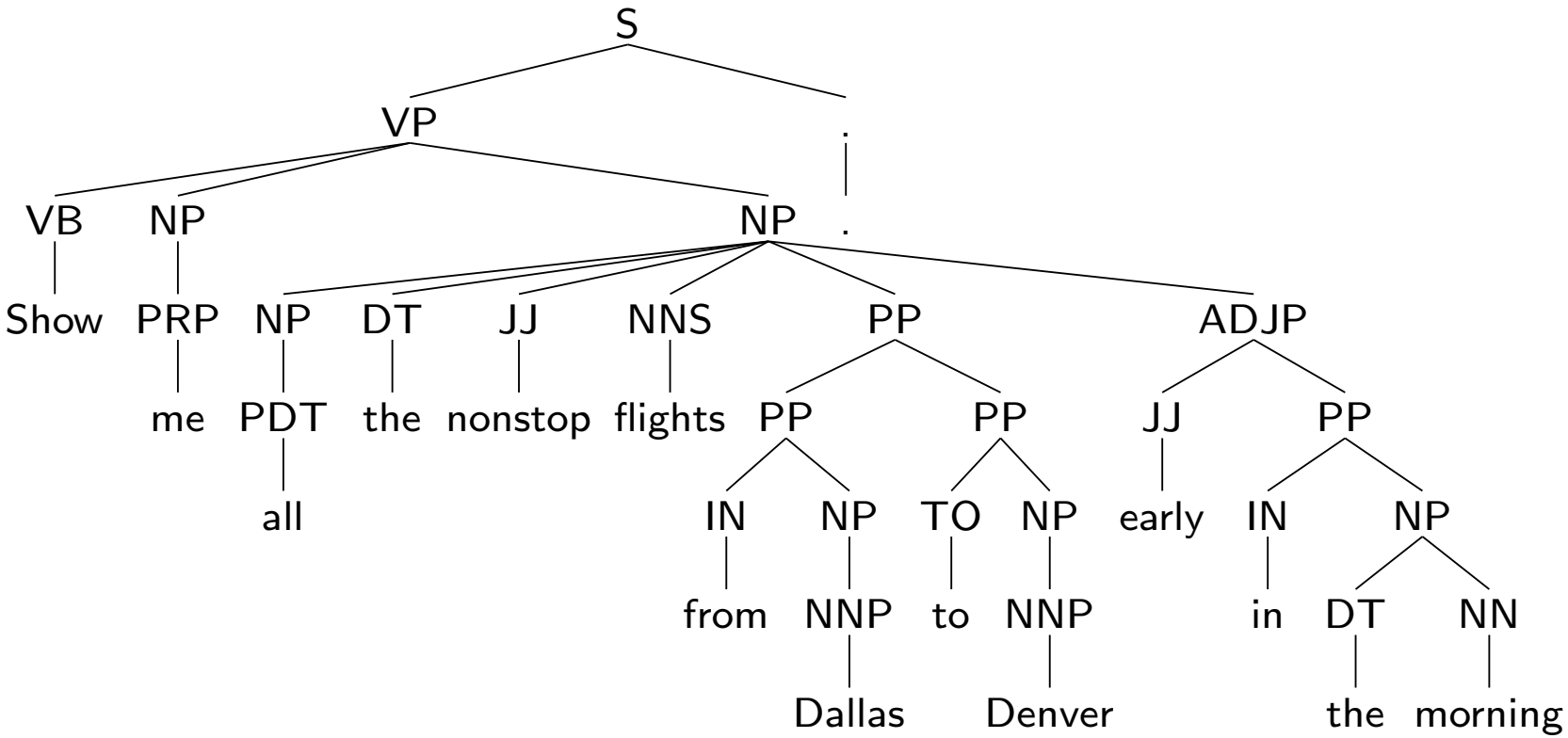


Learning in statistical paradigm

- 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

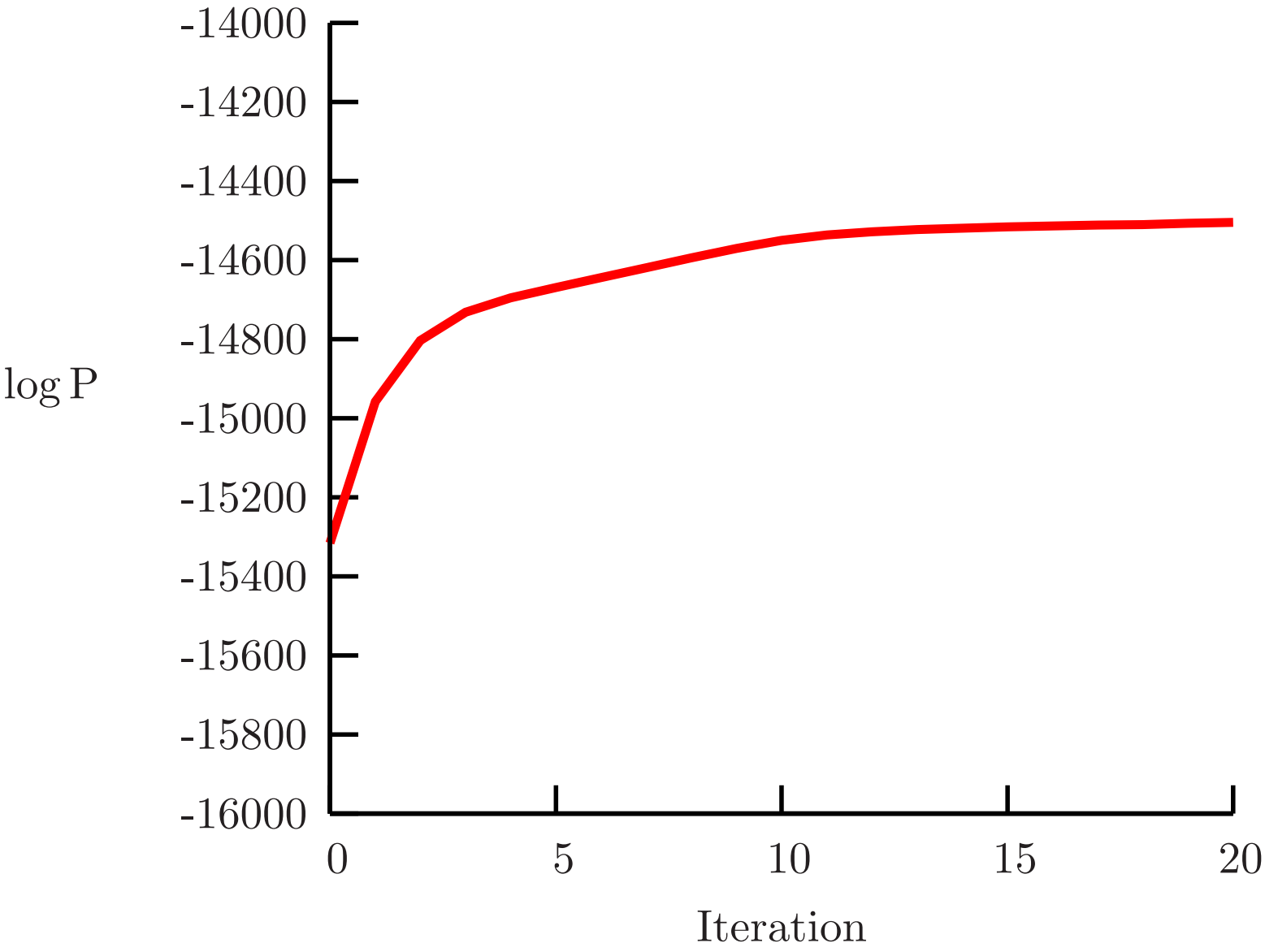
- ATIS treebank consists of 1,300 hand-constructed parse trees
- ignore the words (in this experiment)
- about 1,000 PCFG rules are needed to build these trees



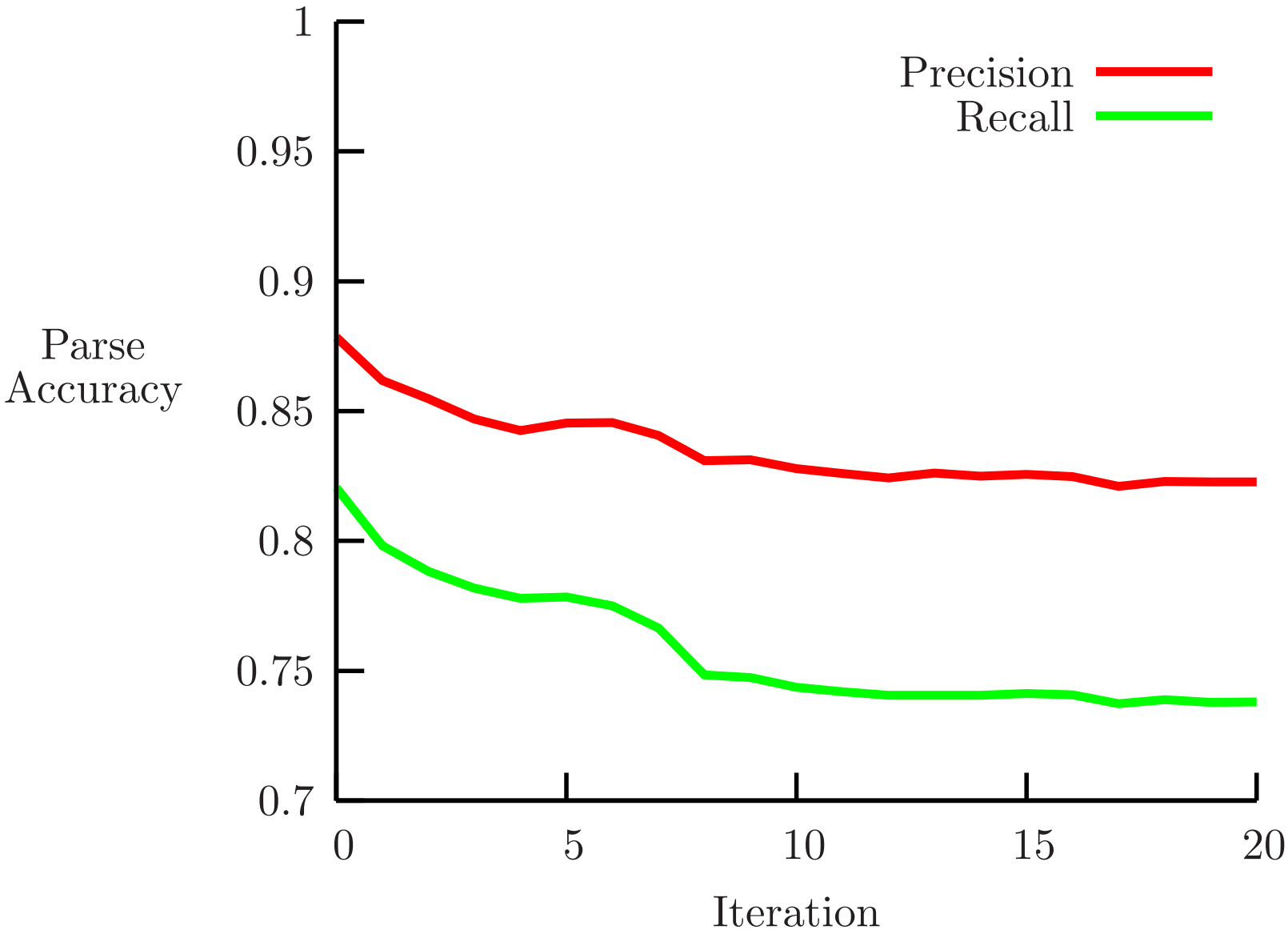
Experiments with EM

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 *grammars are defective*
  - real language has dependencies that these PCFGs can't capture
- How can information about the *marginal distribution of strings* $\mathrm{P}(w)$ provide information about the *conditional distribution of parses given strings* $\mathrm{P}(\psi|w)$?
  - need additional *linking assumptions* about the relationship between parses and strings
- . . . but no one really knows.

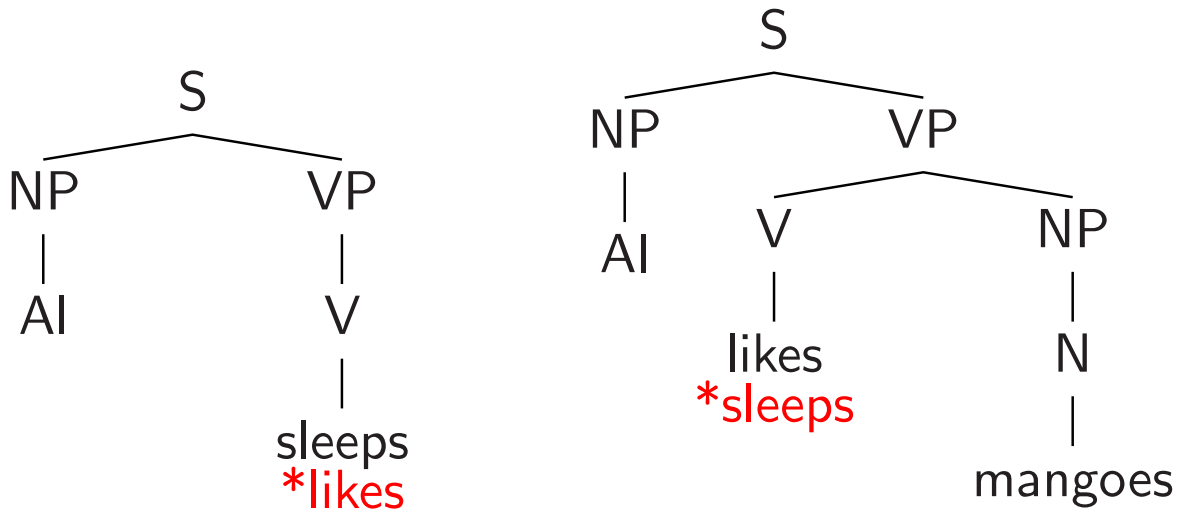
Talk outline

- 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.

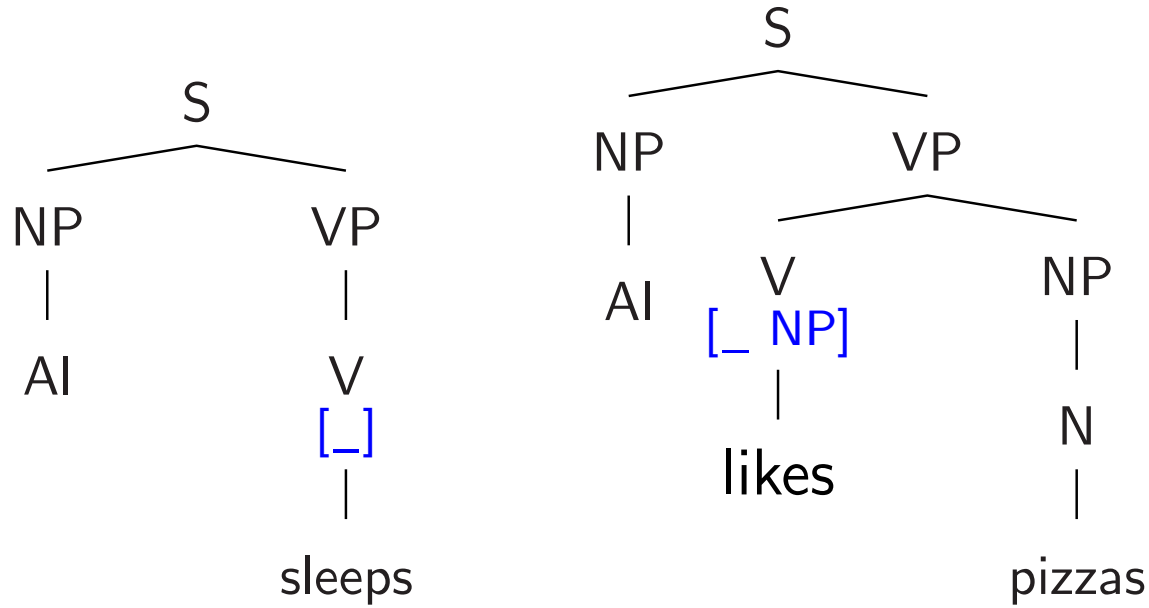
$R_3 = \{\mathsf{VP} \rightarrow \mathsf{V}, \mathsf{VP} \rightarrow \mathsf{V\,NP}, \mathsf{V} \rightarrow \mathsf{sleeps}, \mathsf{V} \rightarrow \mathsf{likes}, \ldots\}$



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.

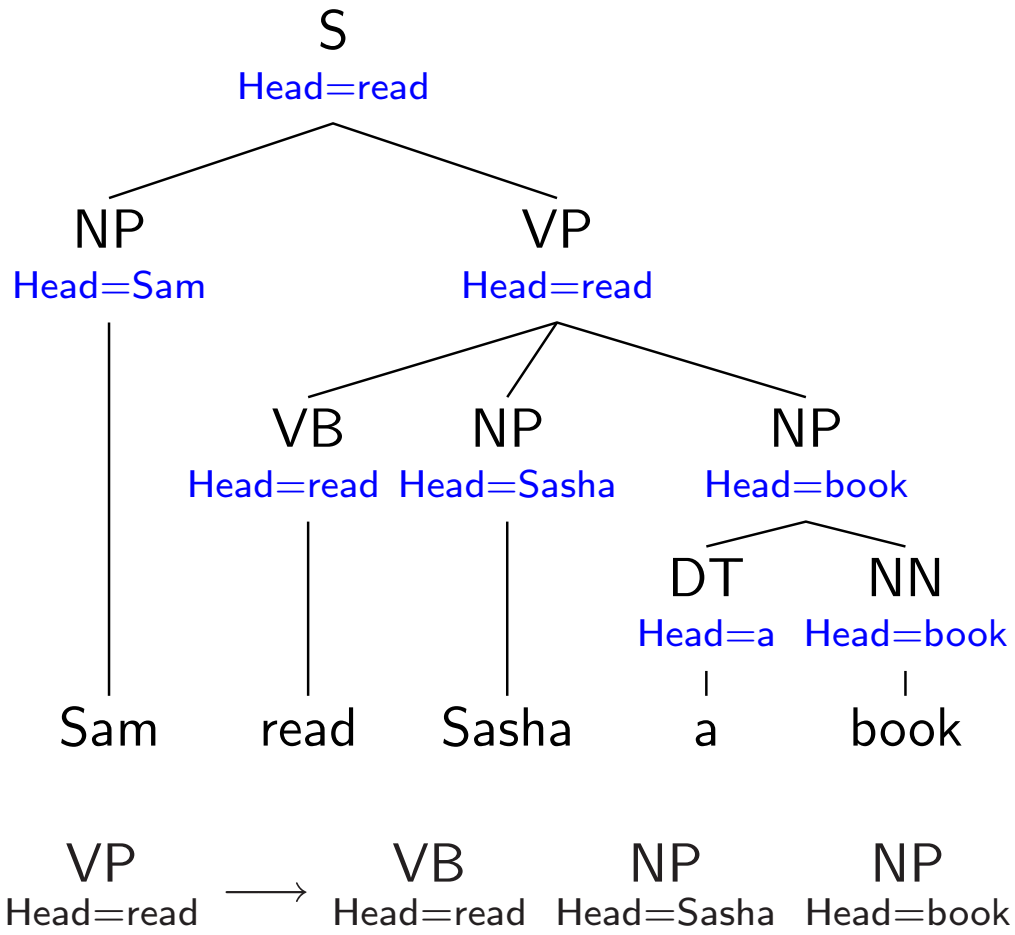


$R_4 = \{\mathsf{VP} \rightarrow \mathsf{V}_{[\_]}, \mathsf{VP} \rightarrow \mathsf{V}_{[\_\ \mathsf{NP}]}\ \mathsf{NP}, \mathsf{V}_{[\_]} \rightarrow \mathsf{sleeps}, \mathsf{V}_{[\_\ \mathsf{NP}]} \rightarrow \mathsf{likes}, \ldots\}$

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

As the non-terminal states are split, *sparse data* becomes a big problem (essential to *generalize* beyond the cases seen in training data)

Head to head dependencies and bilexical rules



Number of possible rules grows rapidly; with a $\approx 10^4$ word vocabulary there are $\approx 10^8$ possible bilexical rules.

Sparse data is biggest problem with such grammars; must *generalize beyond* training data (*smoothing* and *regularization*)

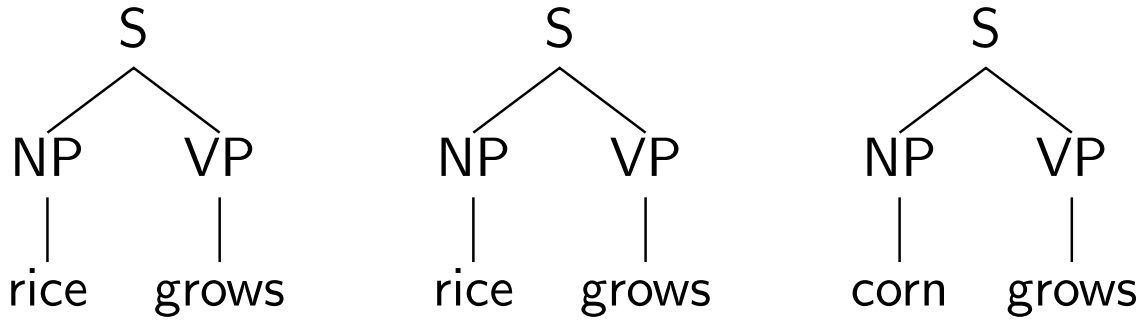
Summary and Conclusion

- Computational linguistics is great fun!
- . . . and maybe will help us understand deep things about language and the mind
- Language possesses a *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 realistic grammars
  - little is known about combining *unsupervised learning* and *smoothing*

Talk outline

- What is computational linguistics?
- Probabilistic grammars
- Identifying phrase structure (parsing)
- Learning phrase structure
- More realistic grammars (part 2)

Estimating PCFGs from visible data

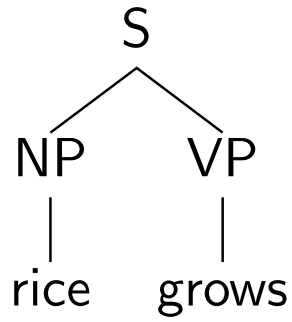


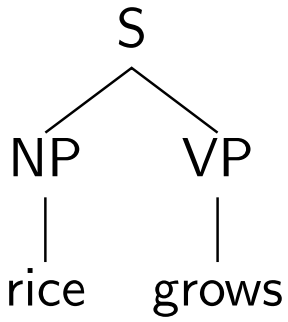
Rule	**Count**	**Rel Freq**
S → NP VP	3	1
NP → rice	2	2/3
NP → corn	1	1/3
VP → grows	3	1

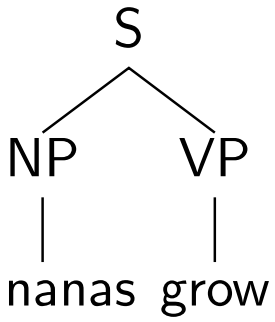
$$\mathrm{P}\left(\begin{array}{c}\mathsf{S}\\ \mathsf{NP}\quad\mathsf{VP}\\ \mathsf{rice}\quad\mathsf{grows}\end{array}\right) = 2/3$$

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

Non-local constraints and PCFG MLE







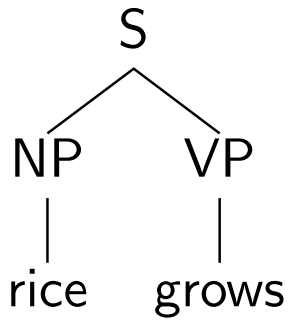
Rule	Count	Rel Freq
S → NP VP	3	1
NP → rice	2	2/3
NP → bananas	1	1/3
VP → grows	2	2/3
VP → grow	1	1/3

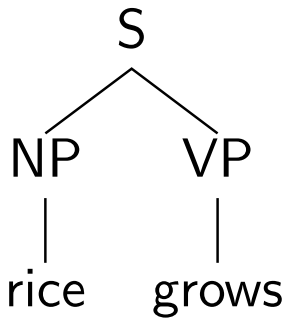
$$\mathrm{P}\left(\begin{array}{c}\text{S}\\ \text{NP}\quad\text{VP}\\ \text{rice}\quad\text{grows}\end{array}\right) = 4/9$$

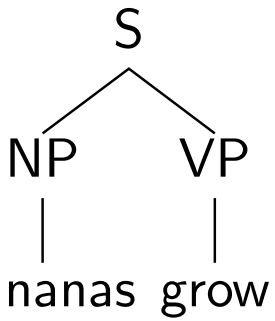
$$\mathrm{P}\left(\begin{array}{c}\text{S}\\ \text{NP}\quad\text{VP}\\ \text{bananas}\quad\text{grow}\end{array}\right) = 1/9$$

partition function $Z = 5/9$

Dividing by partition function Z







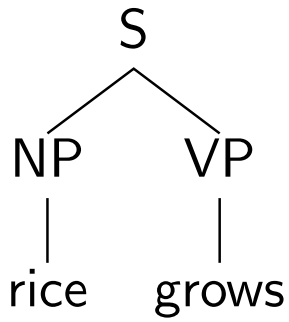
Rule	Count	Rel Freq
S → NP VP	3	1
NP → rice	2	2/3
NP → bananas	1	1/3
VP → grows	2	2/3
VP → grow	1	1/3

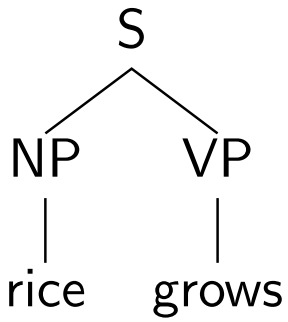
$$\mathrm{P}\left(\begin{array}{c}\mathsf{S}\\ \mathsf{NP}\quad\mathsf{VP}\\ \mathsf{rice}\quad\mathsf{grows}\end{array}\right) = 4/9 \quad 4/5$$

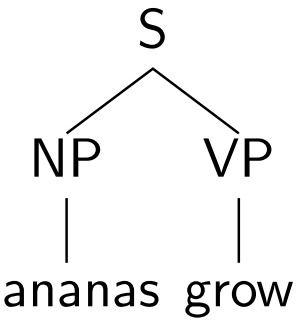
$$\mathrm{P}\left(\begin{array}{c}\mathsf{S}\\ \mathsf{NP}\quad\mathsf{VP}\\ \mathsf{bananas}\quad\mathsf{grow}\end{array}\right) = 1/9 \quad 1/5$$

$$Z = 5/9$$

Other values do better!







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

(Abney 1997)

$$\mathrm{P}\left(\begin{array}{c}\mathrm{S}\\ \mathrm{NP}\quad\mathrm{VP}\\ \mathrm{rice}\quad\mathrm{grows}\end{array}\right) = 2/6 \quad 2/3$$

$$\mathrm{P}\left(\begin{array}{c}\mathrm{S}\\ \mathrm{NP}\quad\mathrm{VP}\\ \mathrm{bananas}\quad\mathrm{grow}\end{array}\right) = 1/6 \quad 1/3$$

$$Z = 3/6$$

Summary so far

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

$\Rightarrow$ relative frequency estimator not MLE

  – Make non-local dependencies local by splitting categories

  $\Rightarrow$ Astronomical number of possible categories

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

Exponential models

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

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

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

PCFGs are exponential models

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

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

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

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

$$f\left(\begin{array}{c} \text{S} \\ \text{NP} \quad \text{VP} \\ \text{rice} \quad \text{grows} \end{array}\right) = [\underbrace{1}_{\text{S}\to\text{NP VP}}, \underbrace{1}_{\text{NP}\to\text{rice}}, \underbrace{0}_{\text{NP}\to\text{bananas}}, \underbrace{1}_{\text{VP}\to\text{grows}}, \underbrace{0}_{\text{VP}\to\text{grow}}]$$

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

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

Advantages of exponential models

- Exponential models are very flexible . . .
- Features f can be *any function of parses* . . .
  - whether a particular structure occurs in a parse
  - conjunctions of prosodic and syntactic structure
- Parses t need not be trees, but *can be anything at all*
  - Feature structures (LFG, HPSG)
- Exponential models are related to other popular models
  - Harmony theory and optimality theory
  - Maxent models

Modeling dependencies

- It's usually difficult to design a PCFG model that captures a particular set of dependencies
  - probability of the tree must be broken down into a product of *conditional probability distributions*
  - non-local dependencies must be expressed in terms of GPSG-style feature passing
- It's easy to make exponential models sensitive to new dependencies
  - add a new feature functions to existing feature functions
  - *figuring out what the right dependencies are is hard, but incorporating them into an exponential model is easy*

MLE of exponential models is hard

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

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

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

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

Conditional ML estimation

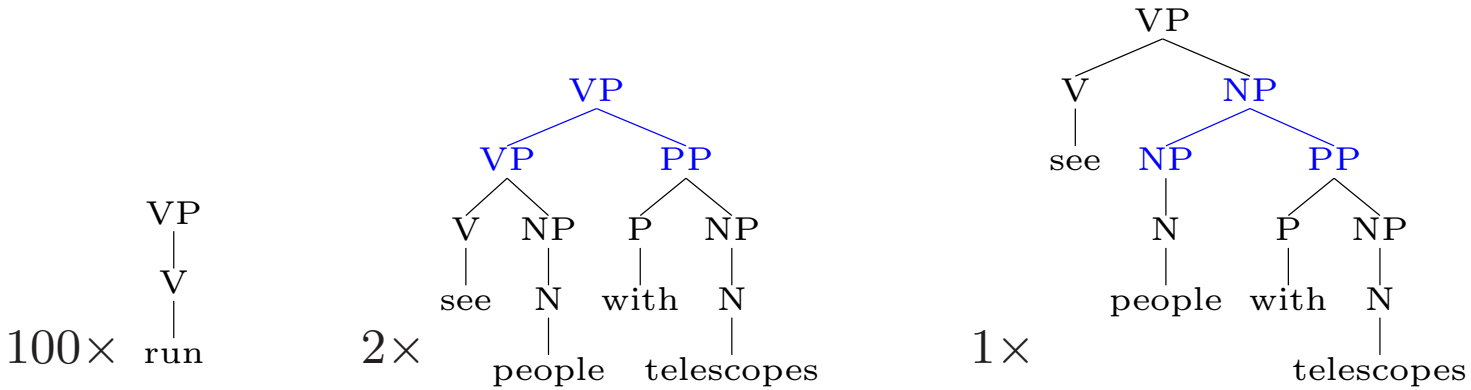
- Conditional ML estimation chooses feature weights to maximize $\mathrm{P}(t_1|s_1) \times \ldots \times \mathrm{P}(t_n|s_n)$, where s_i is string for t_i
  - choose feature weights to make t_i most likely relative to parses $\mathcal{T}(s_i)$ for s_i

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

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

- CMLE "only" involves repeatedly parsing training data
- With "wrong" models, CMLE often produces a more accurate parser than joint MLE

Conditional vs joint MLE



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

Conditional ML estimation

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

- Parser designer specifies *feature functions* $f = (f_1, \ldots, f_m)$
- A parser produces trees $\mathcal{T}(s)$ for each sentence $s \in (s_1, \ldots, s_n)$
- Treebank tells us correct tree $\bar{t}_i \in \mathcal{T}(s_i)$ for sentence s_i
- Feature functions f apply to each tree $t \in \mathcal{T}(s)$, producing feature values $f(t) = (f_1(t), \ldots, f_m(t))$
- MCLE estimates feature weights $\widehat{w} = (\widehat{w}_1, \ldots, \widehat{w}_m)$

Regularization

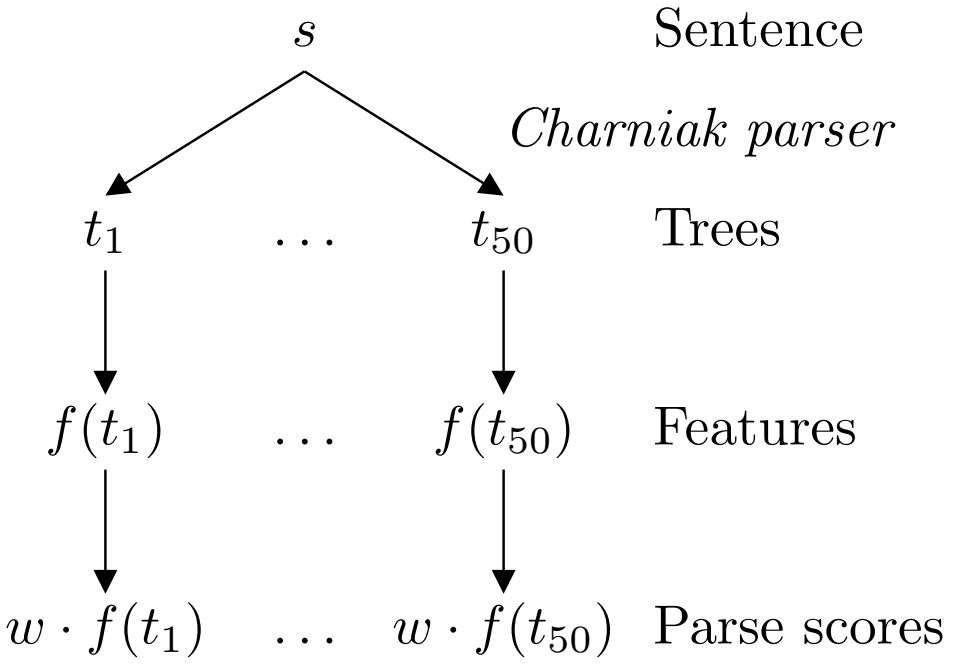
- With a large number of features, exponential models can over-fit
- Regularization: add *bias* term to ensure $\widehat{w}$ is finite and small
- In following experiments, regularizer is a polynomial penalty term

$$\begin{aligned}\widehat{w} &= \operatorname*{argmax}_w \log \sum_{i=1}^{n} \mathrm{P}_w(t_i|s_i) - c\sum_{j=1}^{m} |w_j|^p \\ &= \operatorname*{argmax}_w \sum_{i=1}^{n} \left(\sum_{j=1}^{m} w_j f_j(t_i) - \log Z_w(s) \right) - c\sum_{j=1}^{m} |w_j|^p\end{aligned}$$

- $p = 2$ gives a Gaussian prior.
- We maximize this expression using numerical optimization (Limited Memory Variable Metric)

Coarse-to-fine parsing

- $Z(s)$ is still hard to compute $\Rightarrow$ make $\mathcal{T}(s)$ even smaller!
- Restrict attention to 50-best parses produced by Charniak parser (a good PCFG-based parser)
- Exponential model is trained using CMLE to pick out best parse from Charniak's 50-best parses



Parser evaluation

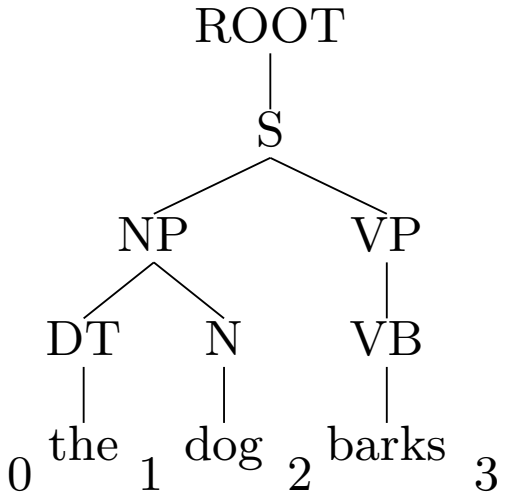
- A node's *edge* is its label and beginning and ending *string positions*
- $E(t)$ is the set of edges associated with a tree t
- If t is a parse tree and $\bar{t}$ is the correct tree, then

  precision $\quad P_{\bar{t}}(t) = |E(t)|/|E(t) \cap E(\bar{t})|$

  recall $\quad R_{\bar{t}}(t) = |E(\bar{t})|/|E(t) \cap E(\bar{t})|$

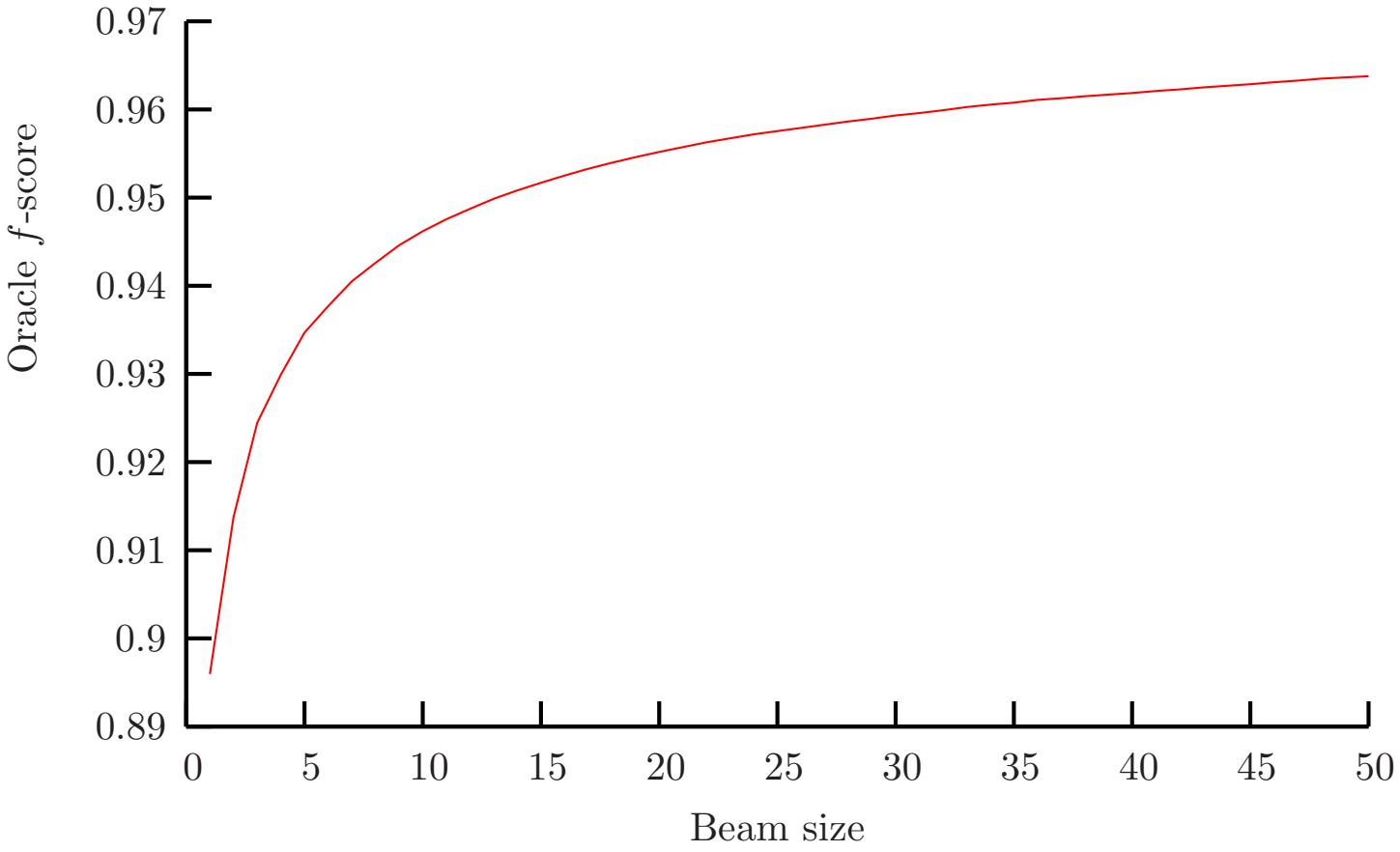
  f-score $\quad F_{\bar{t}}(t) = 2/(P_{\bar{t}}(t)^{-1} + R_{\bar{y}}(t)^{-1})$ (geometric mean of P and R)

Edges		
(0	NP	2)
(2	VP	3)
(0	S	3)



Performance of Charniak parser

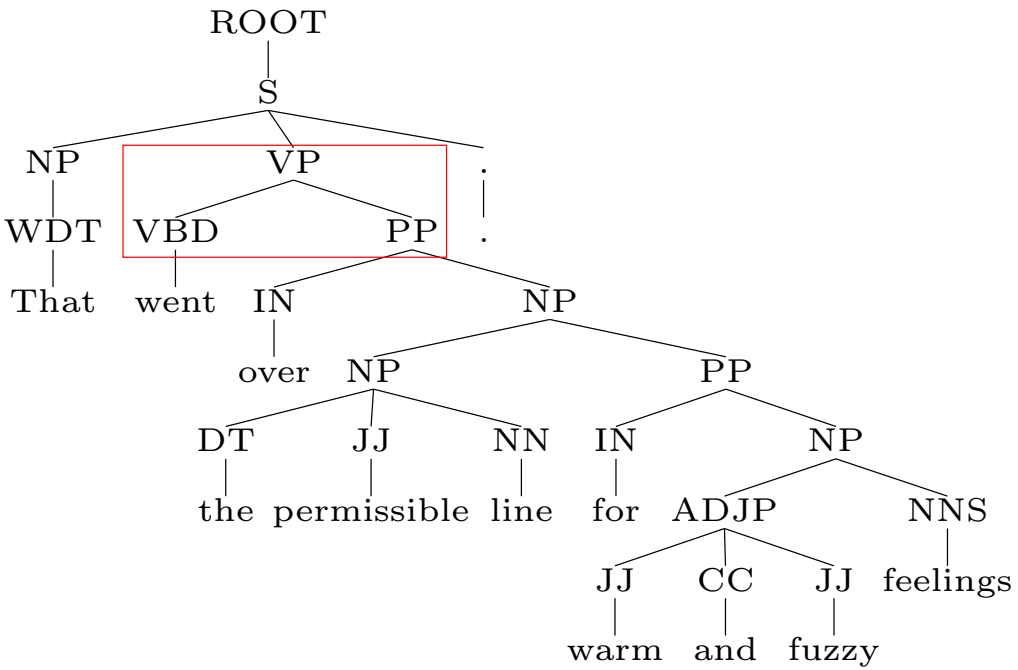
- F-score of Charniak's most probable parse = 0.896 (cross-validated on PTB sections 2-19)
- Oracle f-score of Charniak's 50-best parses = 0.965 (66% redn)



Expt 1: Only "old" features

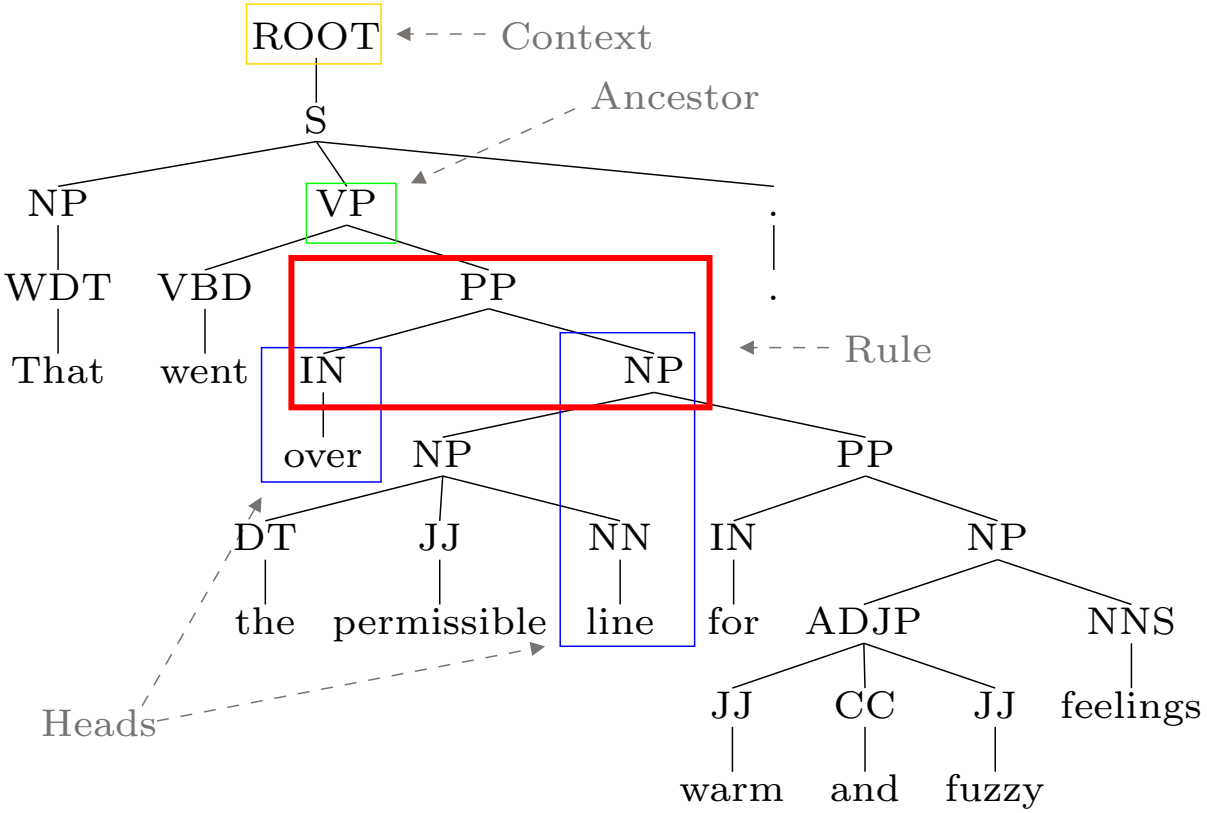
- Features: 1 *log Charniak probability*, 10,124 Rule features
- Charniak's parser already conditions on local trees!
- Feature selection: features must vary on 5 or more sentences
- Results: f-score $= 0.894$; baseline $= 0.890$; $\approx 4\%$ error reduction

$\Rightarrow$ *discriminative training alone can improve accuracy*



Lexicalized and parent-annotated rules

- *Lexicalization* associates each constituent with its head
- *Ancestor annotation* provides a little "vertical context"
- *Context annotation* indicates constructions that only occur in main clause (c.f., Emonds)

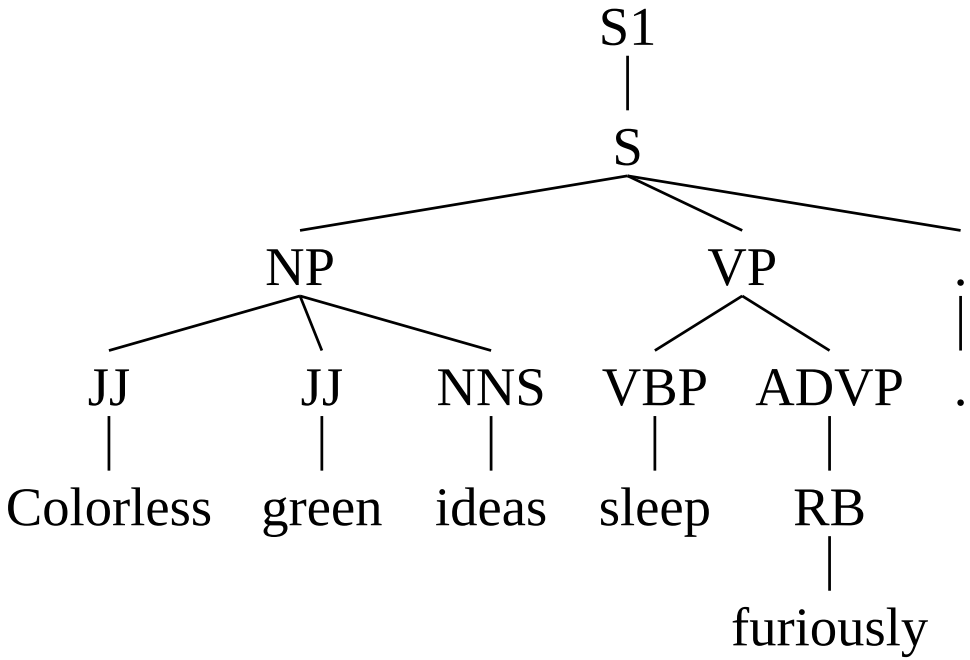


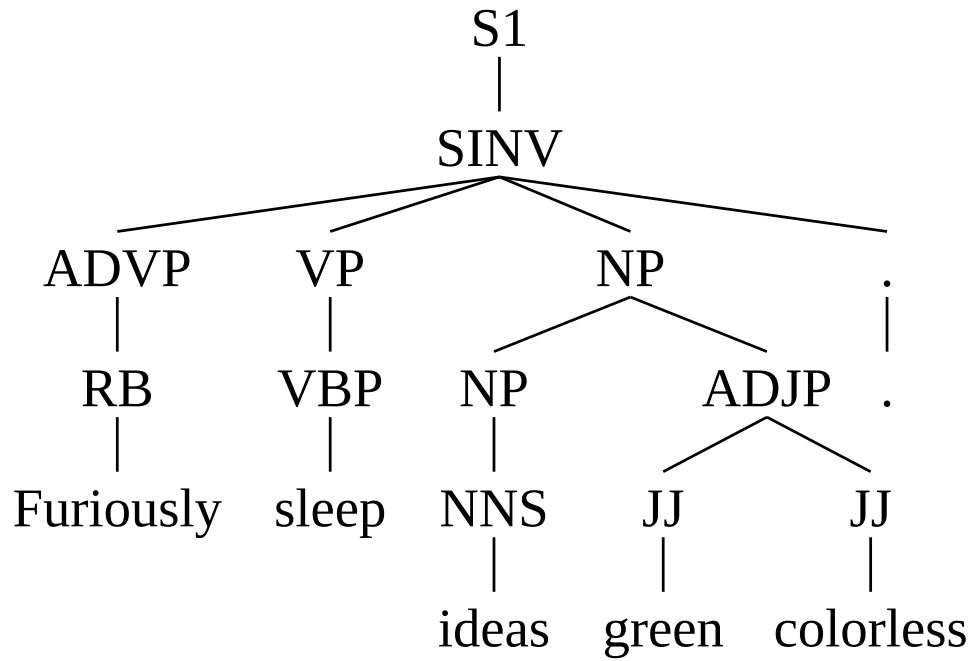
Experimental results with all features

- Features must vary on parses of at least 5 sentences in training data
- In this experiment, 724,550 features
- Gaussian regularization, adjusted via cross-validation on section 23
- *f-score on section 23* = 0.912 (15% error reduction over Charniak parser)

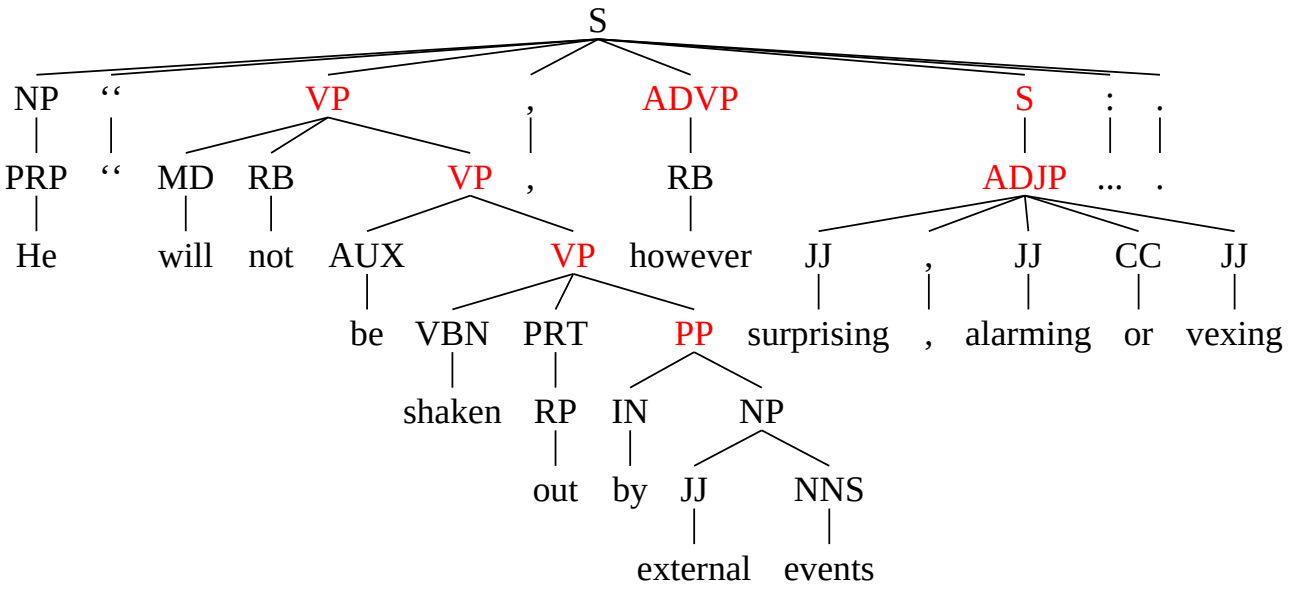
Conclusion

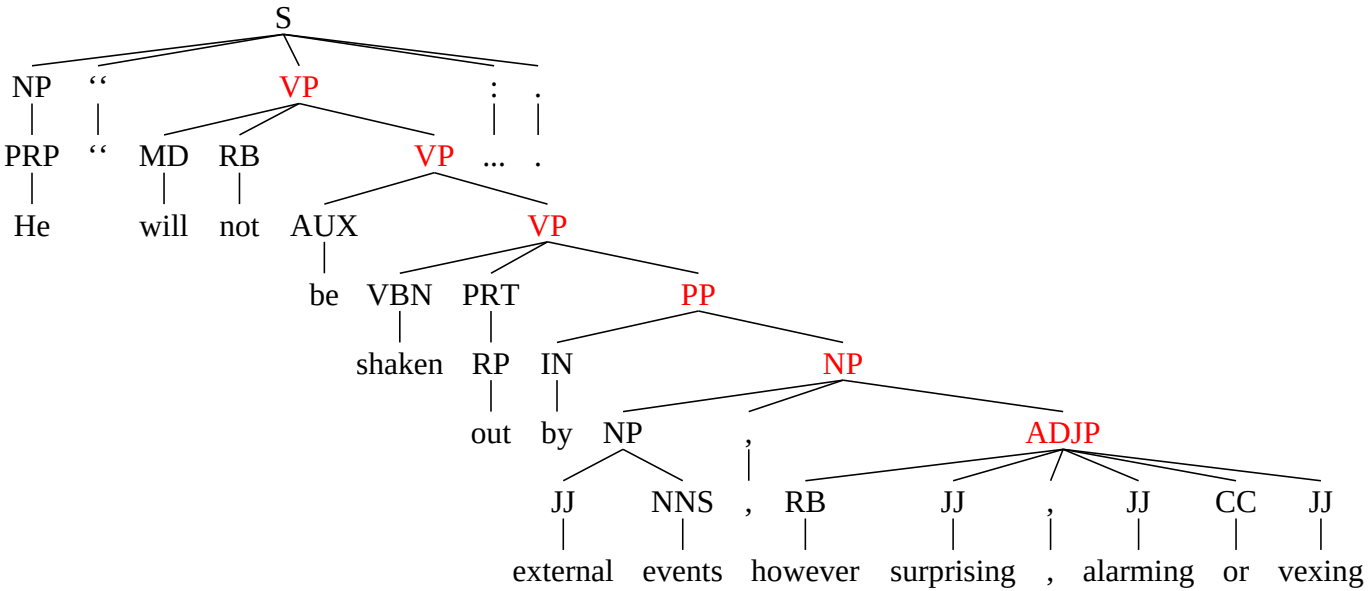
- It's possible to build (moderately) accurate, broad-coverage parsers
- Generative parsing models are easy to estimate, but make questionable independence assumptions
- Exponential models don't assume independence, so it's easy to add new features, but are difficult to estimate
- Coarse-to-fine conditional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

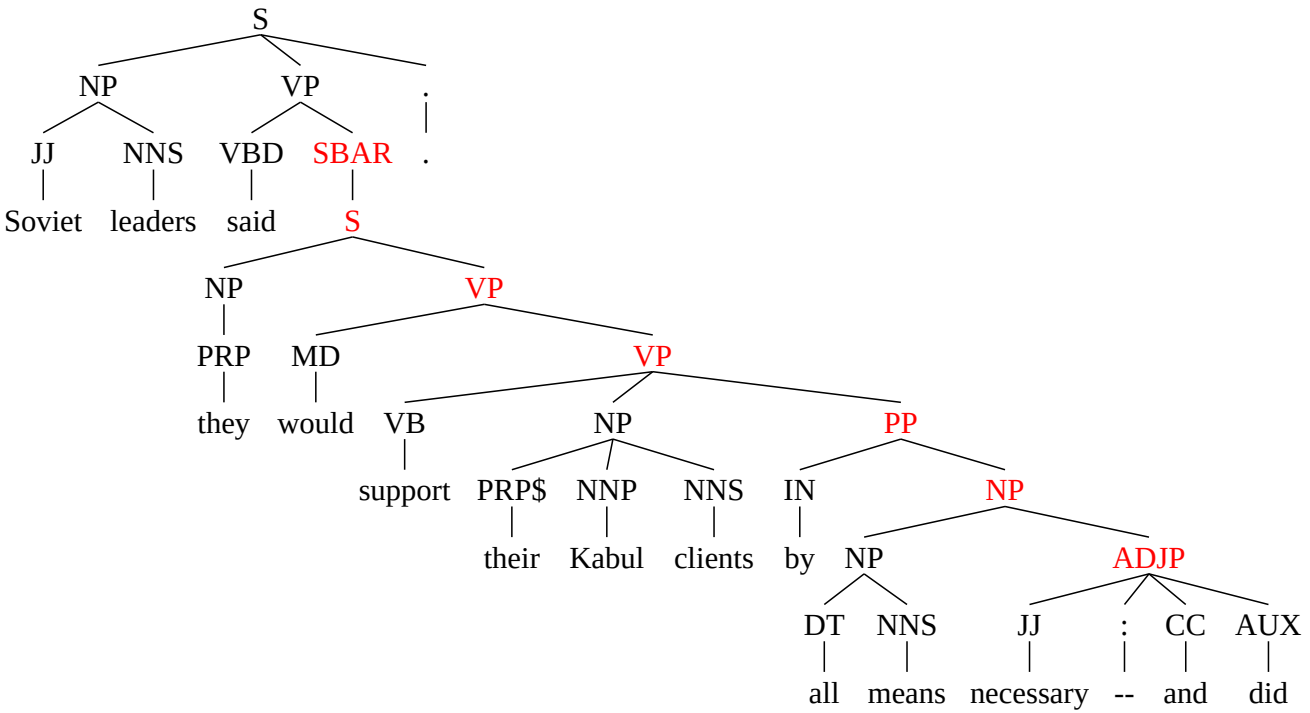

S1
S
NP
VP
.
JJ
JJ
NNS
VBP
ADVP
.
Colorless
green
ideas
sleep
RB
furiously

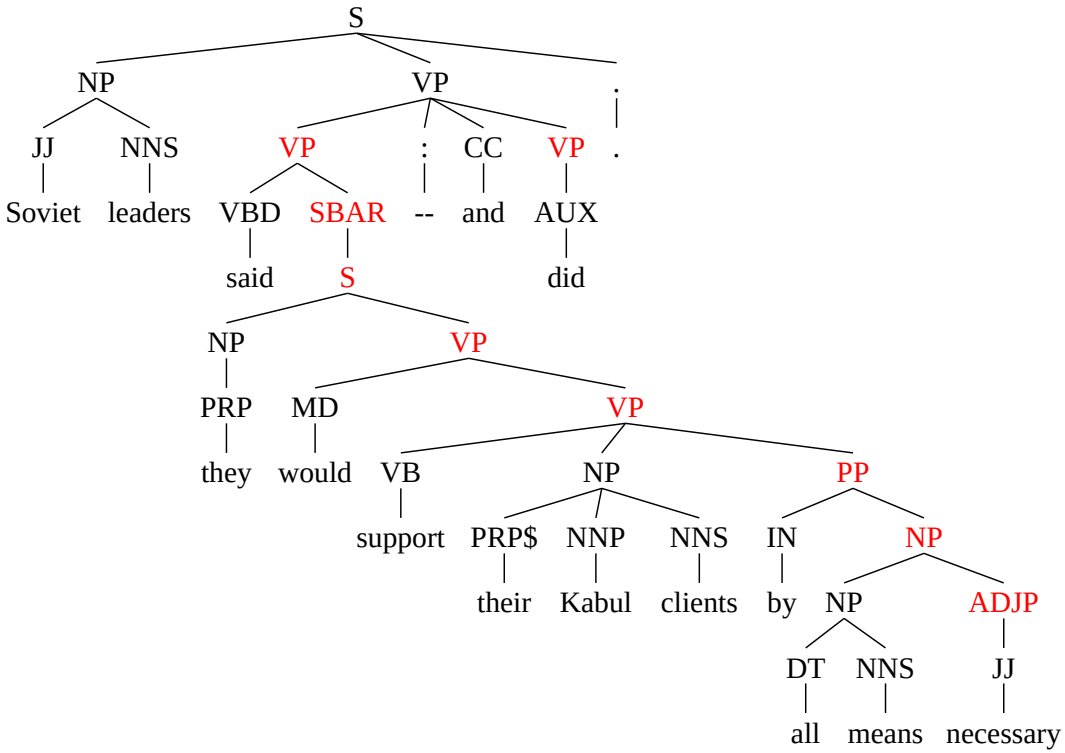

S1
SINV
ADVP
VP
NP
.
RB
VBP
NP
ADJP
.
Furiously
sleep
NNS
JJ
JJ
ideas
green
colorless

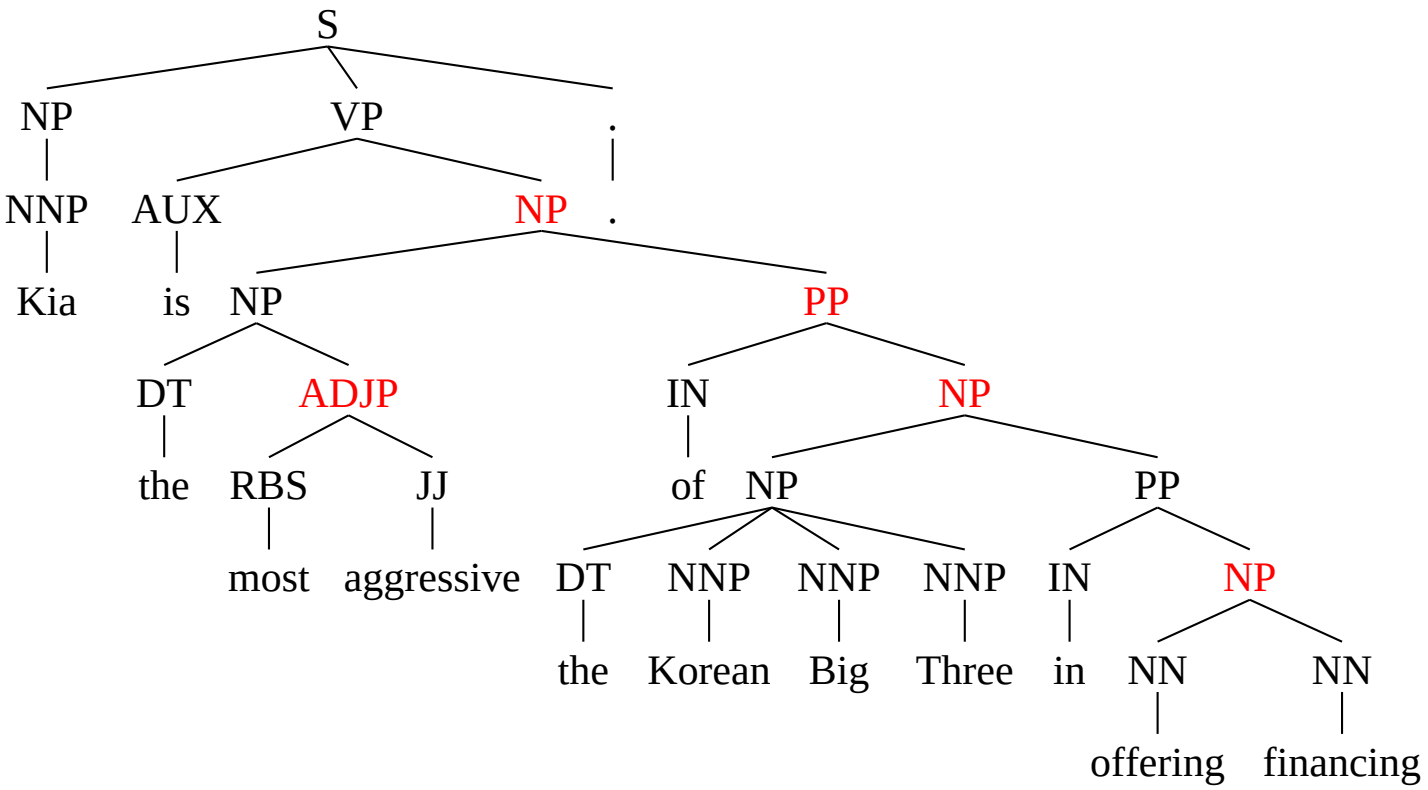
Sample parser errors

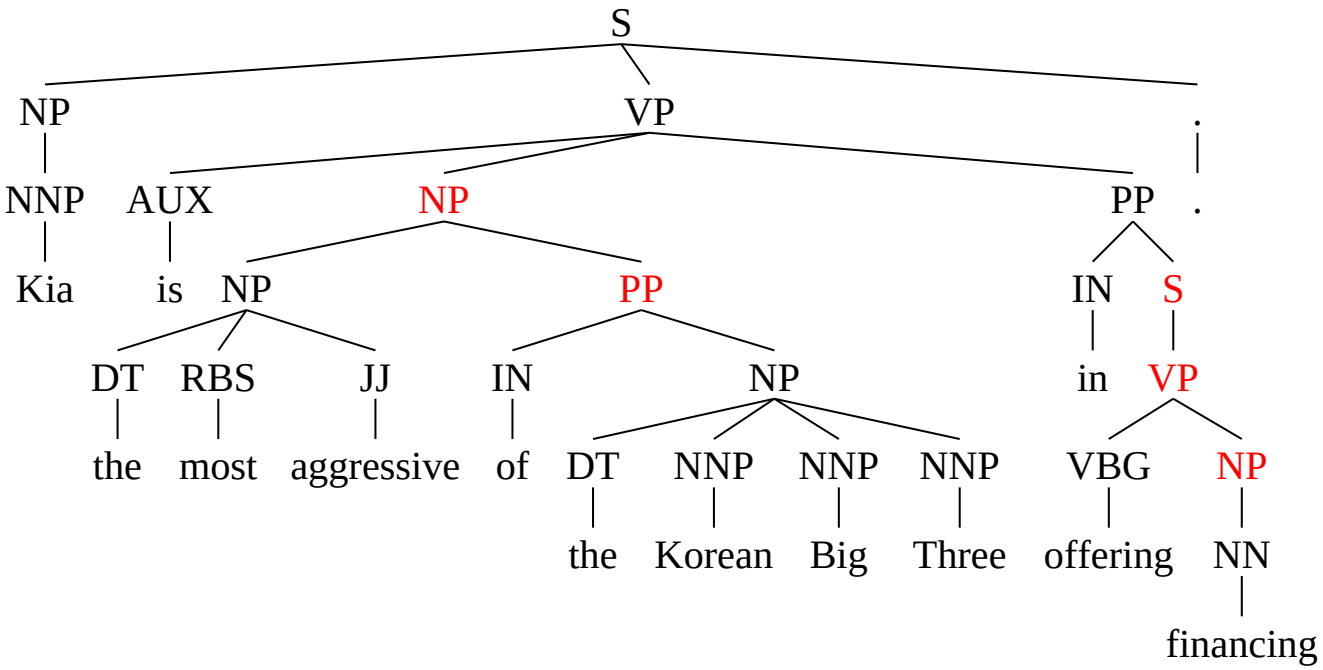


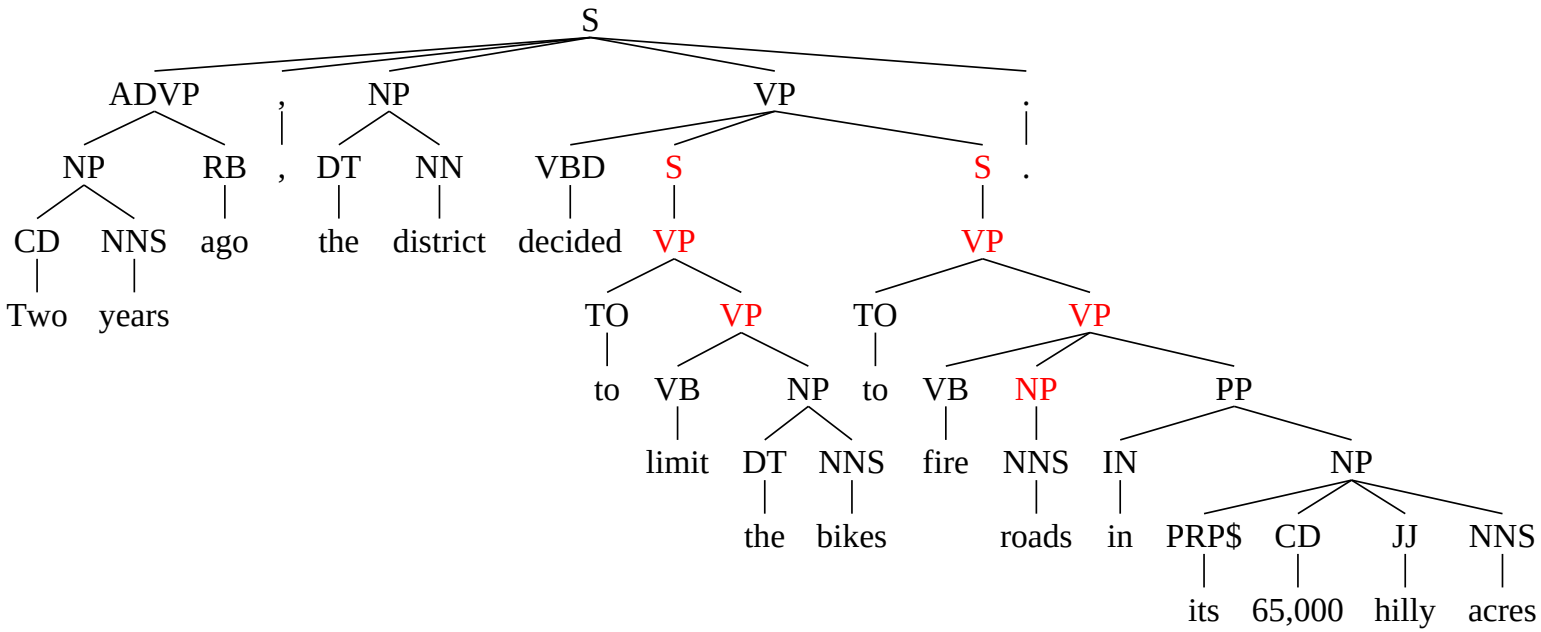


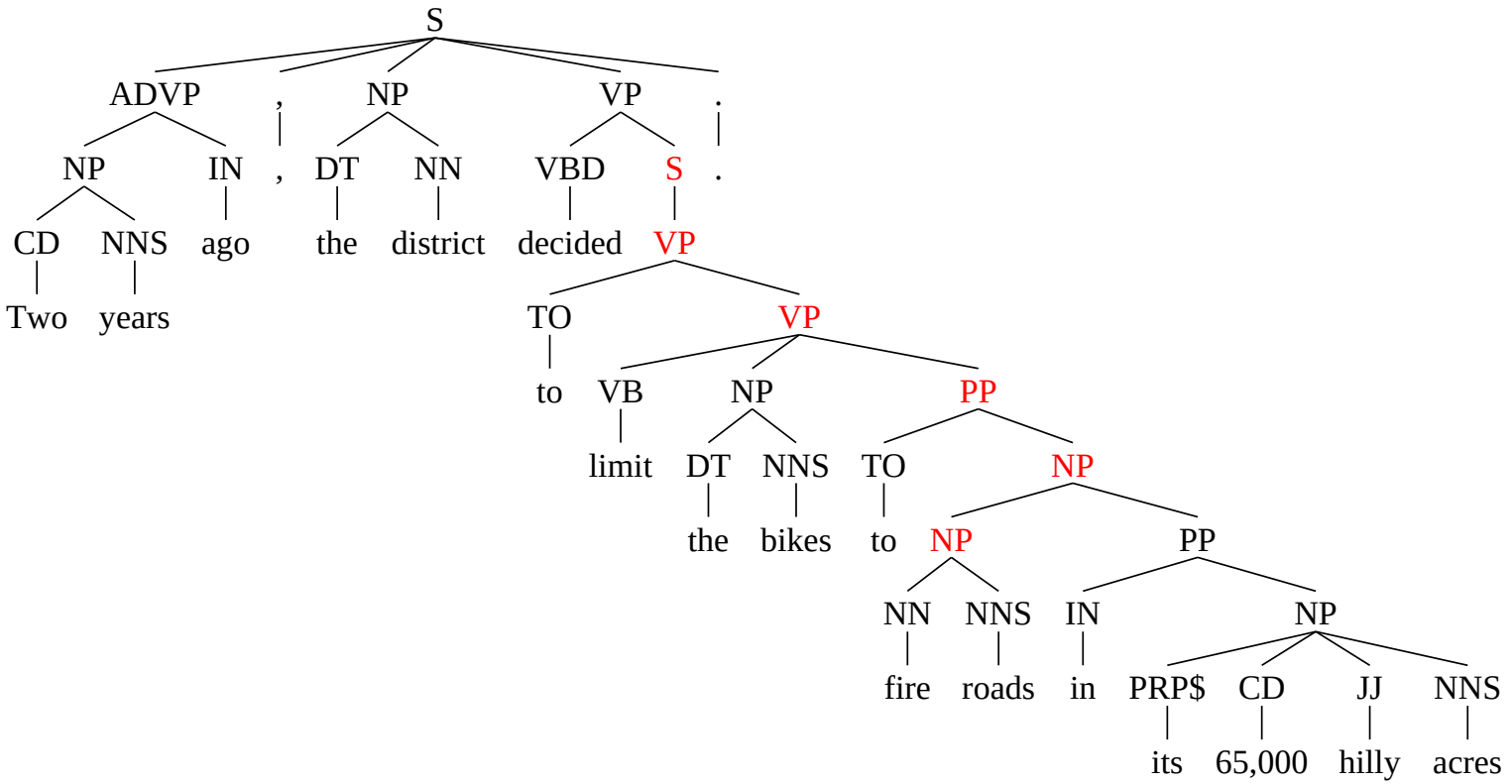

S
NP
VP
.
JJ
NNS
VBD
SBAR
.
Soviet
leaders
said
S
NP
VP
PRP
MD
VP
they
would
VB
NP
PP
support
PRP$
NNP
NNS
IN
NP
their
Kabul
clients
by
NP
ADJP
DT
NNS
JJ
:
CC
AUX
all
means
necessary
--
and
did

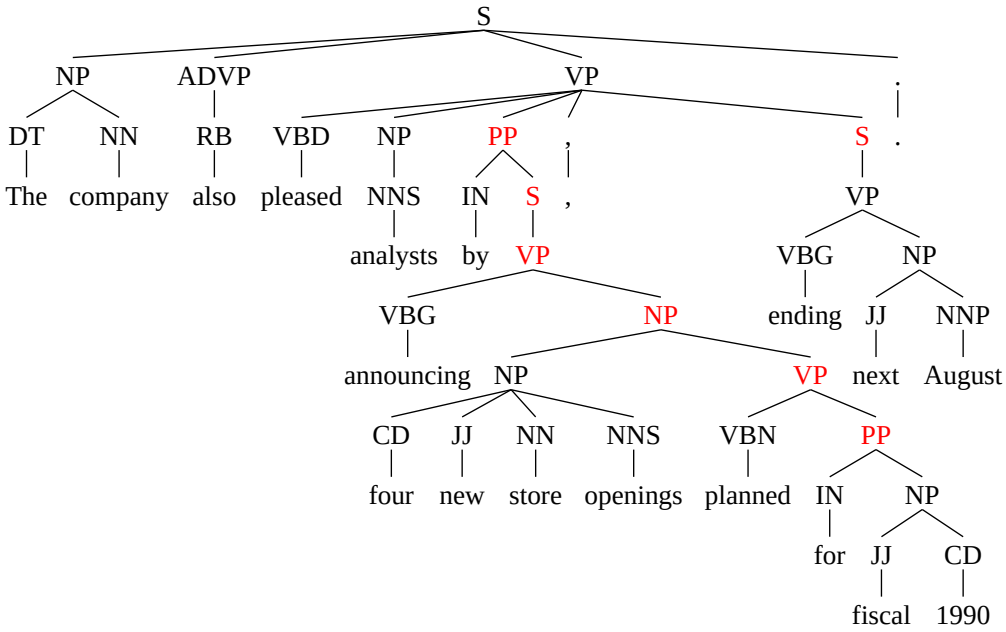

S
NP
VP
.
JJ
NNS
VP
:
CC
VP
.
Soviet
leaders
VBD
SBAR
--
and
AUX
said
S
did
NP
VP
PRP
MD
VP
they
would
VB
NP
PP
support
PRP$
NNP
NNS
IN
NP
their
Kabul
clients
by
NP
ADJP
DT
NNS
JJ
all
means
necessary

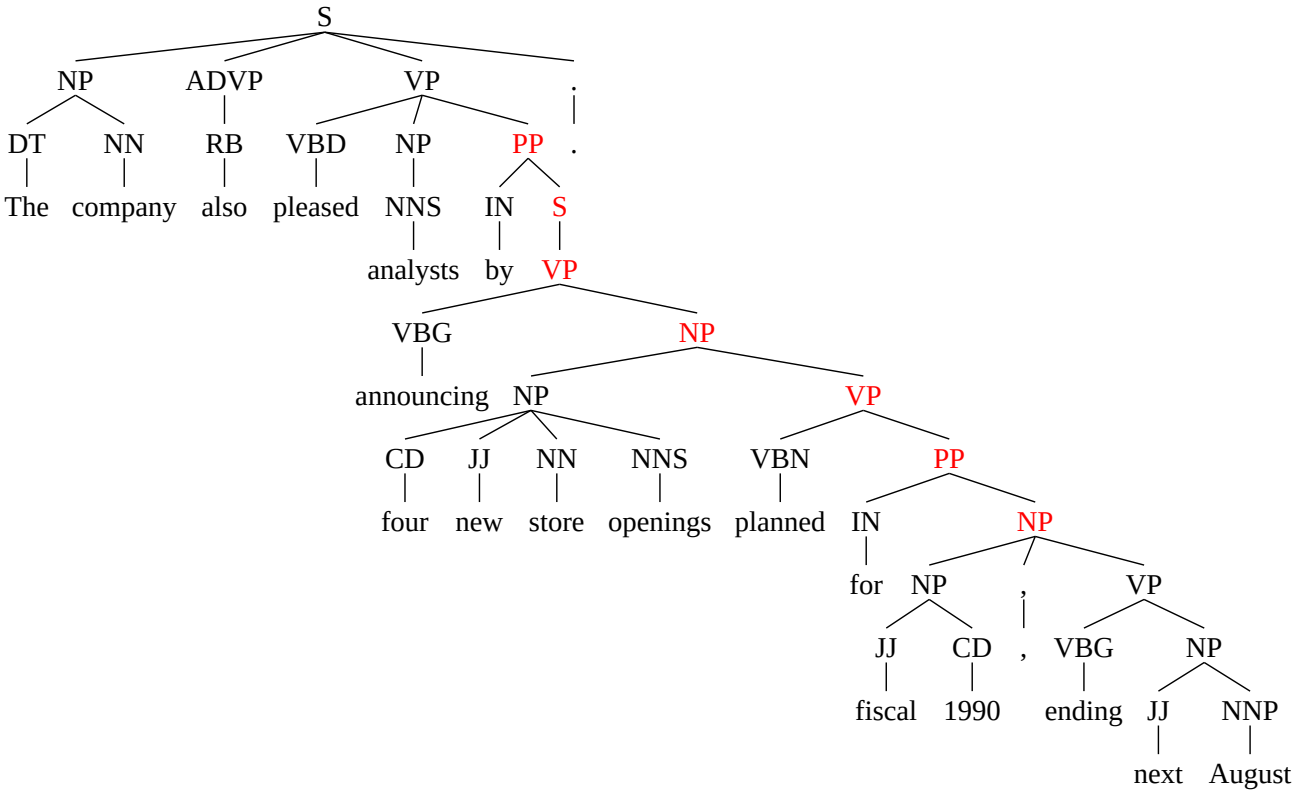

S
NP
VP
.
NNP
AUX
NP
.
Kia
is
NP
PP
DT
ADJP
IN
NP
the
RBS
JJ
of
NP
PP
most
aggressive
DT
NNP
NNP
NNP
IN
NP
the
Korean
Big
Three
in
NN
NN
offering
financing

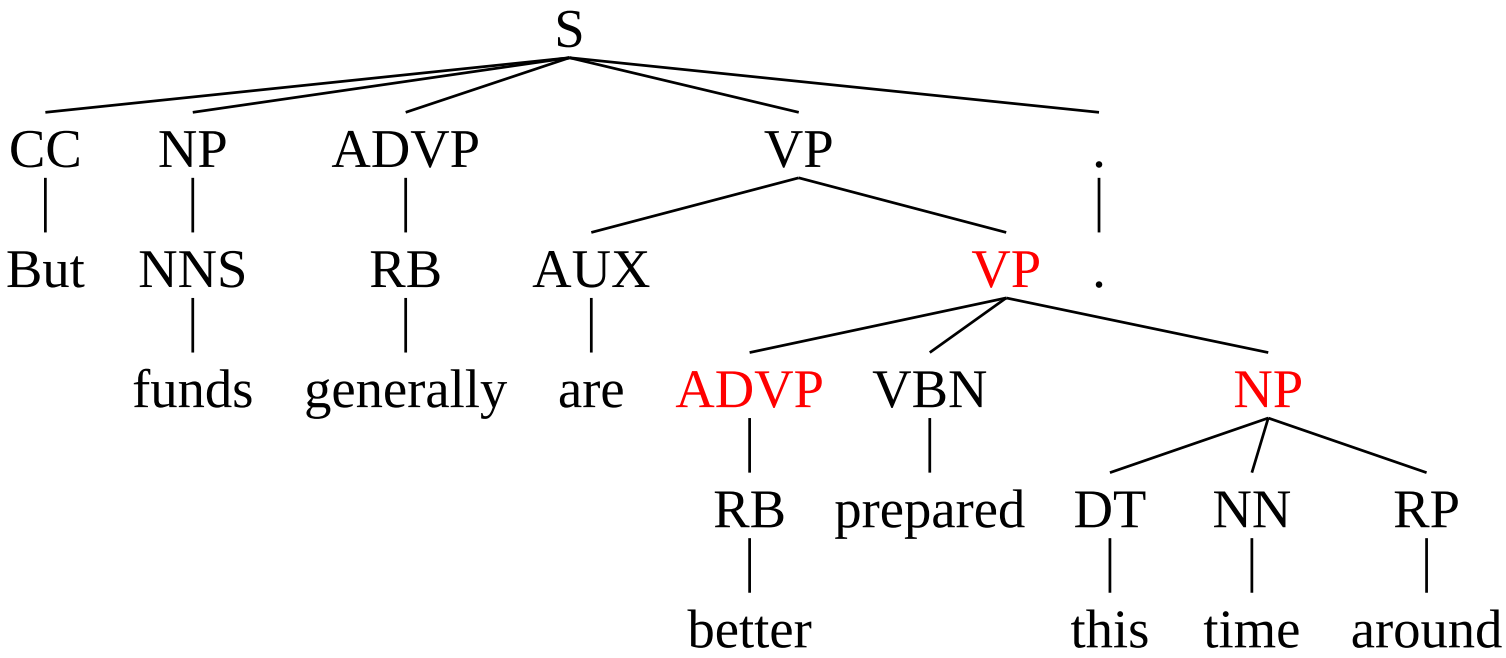

S
NP
VP
.
NNP
AUX
NP
PP
.
Kia
is
NP
PP
IN
S
DT
RBS
JJ
IN
NP
in
VP
the
most
aggressive
of
DT
NNP
NNP
NNP
VBG
NP
the
Korean
Big
Three
offering
NN
financing

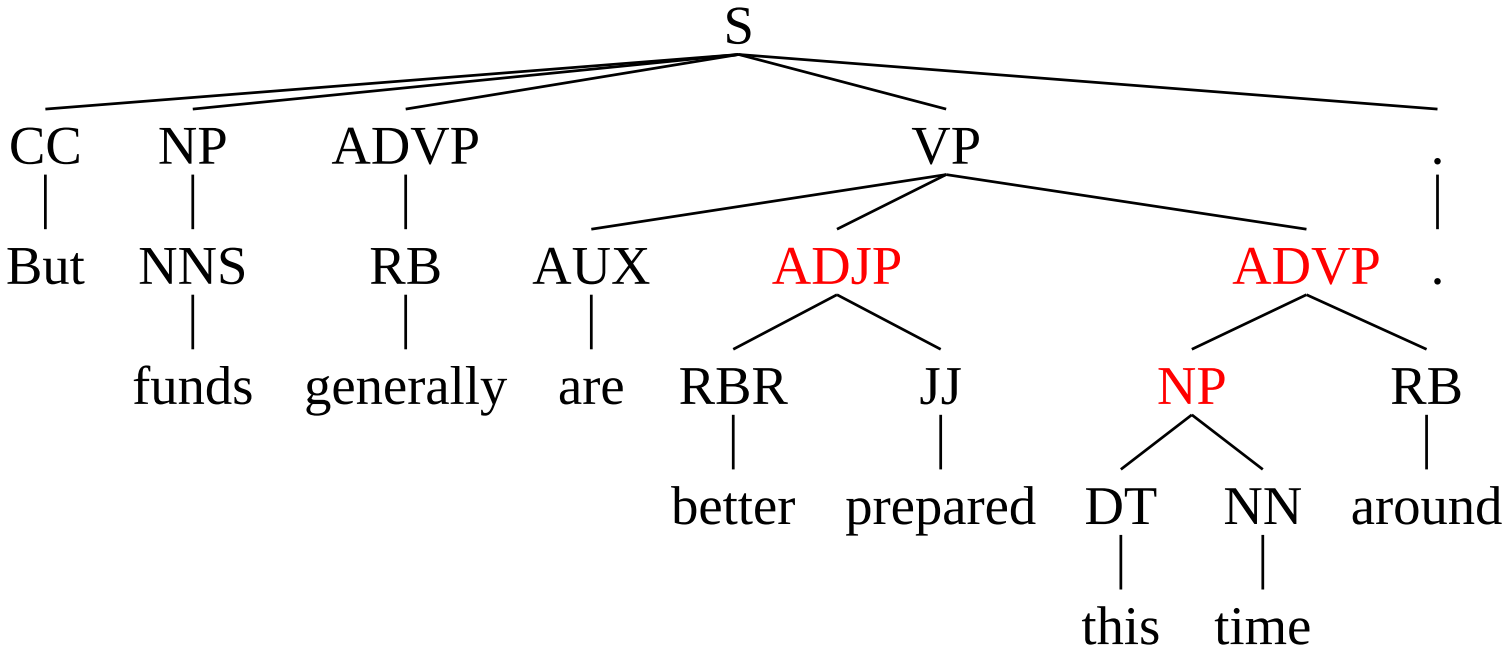

S
ADVP , NP VP .
NP RB , DT NN VBD S S .
CD NNS ago the district decided VP VP
Two years TO VP TO VP
to VB NP to VB NP PP
limit DT NNS fire NNS IN NP
the bikes roads in PRP$ CD JJ NNS
its 65,000 hilly acres

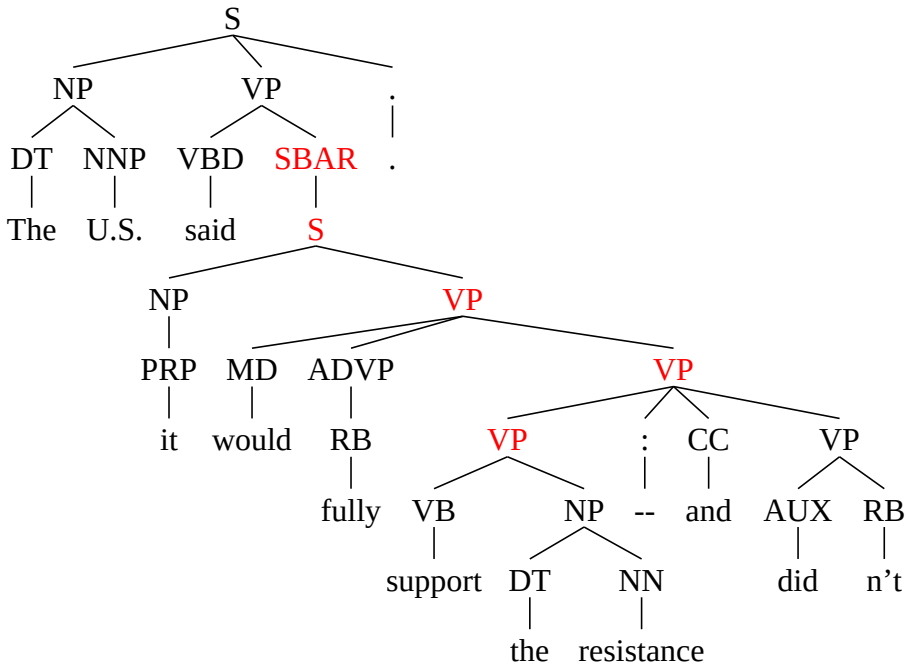

S
ADVP , NP VP .
NP IN , DT NN VBD S .
CD NNS ago the district decided VP
Two years TO VP
to VB NP PP
limit DT NNS TO NP
the bikes to NP PP
NN NNS IN NP
fire roads in PRP$ CD JJ NNS
its 65,000 hilly acres

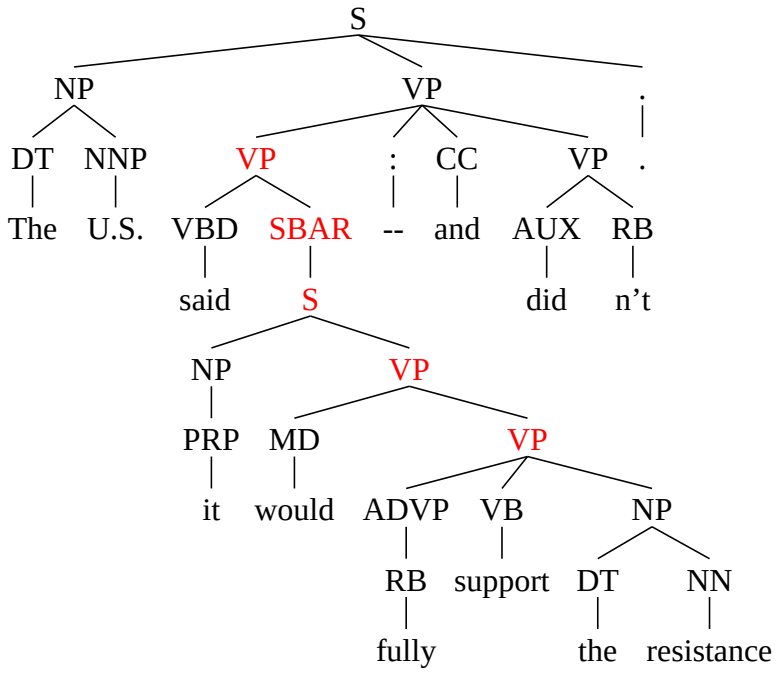

S
NP
ADVP
VP
.
DT
NN
RB
VBD
NP
PP
,
S
.
The
company
also
pleased
NNS
IN
S
,
VP
analysts
by
VP
VBG
NP
VBG
NP
ending
JJ
NNP
announcing
NP
VP
next
August
CD
JJ
NN
NNS
VBN
PP
four
new
store
openings
planned
IN
NP
for
JJ
CD
fiscal
1990


S
NP
ADVP
VP
.
DT
NN
RB
VBD
NP
PP
.
The
company
also
pleased
NNS
IN
S
analysts
by
VP
VBG
NP
announcing
NP
VP
CD
JJ
NN
NNS
VBN
PP
four
new
store
openings
planned
IN
NP
for
NP
,
VP
JJ
CD
,
VBG
NP
fiscal
1990
ending
JJ
NNP
next
August


S
CC
NP
ADVP
VP
.
But
NNS
RB
AUX
VP
.
funds
generally
are
ADVP
VBN
NP
RB
prepared
DT
NN
RP
better
this
time
around


S
CC
NP
ADVP
VP
.
But
NNS
RB
AUX
ADJP
ADVP
.
funds
generally
are
RBR
JJ
NP
RB
better
prepared
DT
NN
around
this
time


S
NP
VP
.
DT
NNP
VBD
SBAR
.
The
U.S.
said
S
NP
VP
PRP
MD
ADVP
VP
it
would
RB
VP
:
CC
VP
fully
VB
NP
--
and
AUX
RB
support
DT
NN
did
n’t
the
resistance


S
NP
VP
.
DT
NNP
VP
:
CC
VP
.
The
U.S.
VBD
SBAR
--
and
AUX
RB
said
S
did
n’t
NP
VP
PRP
MD
VP
it
would
ADVP
VB
NP
RB
support
DT
NN
fully
the
resistance