Statistical Natural Language Parsing

Mausam

(Based on slides of Michael Collins, Dan Jurafsky, Dan Klein, Chris Manning, Ray Mooney, Luke Zettlemoyer)
Two views of linguistic structure:
1. Constituency (phrase structure)

- Phrase structure organizes words into nested constituents.
- How do we know what is a constituent? (Not that linguists don’t argue about some cases.)
  - Distribution: a constituent behaves as a unit that can appear in different places:
    - John talked [to the children] [about drugs].
    - John talked [about drugs] [to the children].
    - *John talked drugs to the children about
  - Substitution/expansion/pro-forms:
    - I sat [on the box/right on top of the box/there].
  - Coordination, regular internal structure, no intrusion, fragments, semantics, ...
Analysts said that Mr. Stronach wants to resume a more influential role in the company.
Two views of linguistic structure:
2. Dependency structure

- Dependency structure shows which words depend on (modify or are arguments of) which other words.
Why Parse?

- Part of speech information
- Phrase information
- Useful relationships

⇒ “the burglar” is the subject of “robbed”
The rise of annotated data: The Penn Treebank

[Marcus et al. 1993, *Computational Linguistics*]
### Penn Treebank Non-terminals

**Table 1.2.** The Penn Treebank syntactic tagset

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADJP</td>
<td>Adjective phrase</td>
</tr>
<tr>
<td>ADVP</td>
<td>Adverb phrase</td>
</tr>
<tr>
<td>NP</td>
<td>Noun phrase</td>
</tr>
<tr>
<td>PP</td>
<td>Prepositional phrase</td>
</tr>
<tr>
<td>S</td>
<td>Simple declarative clause</td>
</tr>
<tr>
<td>SBAR</td>
<td>Subordinate clause</td>
</tr>
<tr>
<td>SBARQ</td>
<td>Direct question introduced by <em>wh</em>-element</td>
</tr>
<tr>
<td>SINV</td>
<td>Declarative sentence with subject-aux inversion</td>
</tr>
<tr>
<td>SQ</td>
<td>Yes/no questions and subconstituent of SBARQ excluding <em>wh</em>-element</td>
</tr>
<tr>
<td>VP</td>
<td>Verb phrase</td>
</tr>
<tr>
<td>WHADVP</td>
<td>Wh-adverb phrase</td>
</tr>
<tr>
<td>WHNP</td>
<td>Wh-noun phrase</td>
</tr>
<tr>
<td>WHPP</td>
<td>Wh-prepositional phrase</td>
</tr>
<tr>
<td>X</td>
<td>Constituent of unknown or uncertain category</td>
</tr>
<tr>
<td>*</td>
<td>“Understood” subject of infinitive or imperative</td>
</tr>
<tr>
<td>0</td>
<td>Zero variant of <em>that</em> in subordinate clauses</td>
</tr>
<tr>
<td>T</td>
<td>Trace of <em>wh</em>-Constituent</td>
</tr>
</tbody>
</table>
The rise of annotated data

• Starting off, building a treebank seems a lot slower and less useful than building a grammar

• But a treebank gives us many things
  • Reusability of the labor
    • Many parsers, POS taggers, etc.
    • Valuable resource for linguistics
  • Broad coverage
  • Frequencies and distributional information
  • A way to evaluate systems
Statistical parsing applications

Statistical parsers are now robust and widely used in larger NLP applications:

- High precision question answering [Pasca and Harabagiu SIGIR 2001]
- Improving biological named entity finding [Finkel et al. JNLPBA 2004]
- Syntactically based sentence compression [Lin and Wilbur 2007]
- Extracting opinions about products [Bloom et al. NAACL 2007]
- Improved interaction in computer games [Gorniak and Roy 2005]
- Helping linguists find data [Resnik et al. BLS 2005]
- Source sentence analysis for machine translation [Xu et al. 2009]
- Relation extraction systems [Fundel et al. Bioinformatics 2006]
Example Application: Machine Translation

- The boy put the tortoise on the rug
- लड़के ने रखा कछुआ ऊपर कालीन
- SVO vs. SOV; preposition vs. post-position
Example Application: Machine Translation

- The boy put the tortoise on the rug
- लड़के ने रखा कछुआ ऊपर कालीन
- SVO vs. SOV; preposition vs. post-position
Example Application: Machine Translation

- The boy put the tortoise on the rug
- लड़के ने रखा कछुआ ऊपर कालीन
- SVO vs. SOV; preposition vs. post-position
Example Application: Machine Translation

- The boy put the tortoise on the rug

- लड़के ने रखा कछुआ ऊपर कालीन

- SVO vs. SOV; preposition vs. post-position
Example Application: Machine Translation

- The boy put the tortoise on the rug
- लड़के ने रखा कछुआ ऊपर कालीन
- SVO vs. SOV; preposition vs. post-position
Pre 1990 (“Classical”) NLP Parsing

• Goes back to Chomsky’s PhD thesis in 1950s
• Wrote symbolic grammar (CFG or often richer) and lexicon
  
  \[
  \begin{align*}
  S & \rightarrow NP \ VP \\
  NP & \rightarrow (DT) \ NN \\
  NP & \rightarrow NN \ NNS \\
  NP & \rightarrow NNP \\
  VP & \rightarrow V \ NP \\
  \end{align*}
  \]

  \[
  \begin{align*}
  \text{NN} & \rightarrow \text{interest} \\
  \text{NNS} & \rightarrow \text{rates} \\
  \text{NNS} & \rightarrow \text{raises} \\
  \text{VBP} & \rightarrow \text{interest} \\
  \text{VBZ} & \rightarrow \text{rates} \\
  \end{align*}
  \]

• Used grammar-proof systems to prove parses from words
• This scaled very badly and didn’t give coverage. For sentence:

  \textit{Fed raises interest rates 0.5\% in effort to control inflation}

  • Minimal grammar: 36 parses
  • Simple 10 rule grammar: 592 parses
  • Real-size broad-coverage grammar: millions of parses
Classical NLP Parsing: The problem and its solution

- Categorical constraints can be added to grammars to limit unlikely/weird parses for sentences
  - But the attempt makes the grammars not robust
    - In traditional systems, commonly 30% of sentences in even an edited text would have no parse.
- A less constrained grammar can parse more sentences
  - But simple sentences end up with ever more parses with no way to choose between them
- We need mechanisms that allow us to find the most likely parse(s) for a sentence
  - Statistical parsing lets us work with very loose grammars that admit millions of parses for sentences but still quickly find the best parse(s)
Context Free Grammars and Ambiguities
Context-Free Grammars

Hopcroft and Ullman, 1979

A context free grammar \( G = (N, \Sigma, R, S) \) where:

- \( N \) is a set of non-terminal symbols
- \( \Sigma \) is a set of terminal symbols
- \( R \) is a set of rules of the form \( X \rightarrow Y_1 Y_2 \ldots Y_n \) for \( n \geq 0, X \in N, Y_i \in (N \cup \Sigma) \)
- \( S \in N \) is a distinguished start symbol
Context-Free Grammars in NLP

• A context free grammar $G$ in NLP = $(N, C, \Sigma, S, L, R)$
  • $\Sigma$ is a set of terminal symbols
  • $C$ is a set of preterminal symbols
  • $N$ is a set of nonterminal symbols
  • $S$ is the start symbol ($S \in N$)
  • $L$ is the lexicon, a set of items of the form $X \rightarrow x$
    • $X \in C$ and $x \in \Sigma$
  • $R$ is the grammar, a set of items of the form $X \rightarrow \gamma$
    • $X \in N$ and $\gamma \in (N \cup C)^*$

• By usual convention, $S$ is the start symbol, but in statistical NLP, we usually have an extra node at the top (ROOT, TOP)
• We usually write $e$ for an empty sequence, rather than nothing
A Context Free Grammar of English

\[ N = \{ S, \text{NP}, \text{VP}, \text{PP}, \text{DT}, \text{Vi}, \text{Vt}, \text{NN}, \text{IN} \} \]
\[ S' = S \]
\[ \Sigma = \{ \text{sleeps, saw, man, woman, telescope, the, with, in} \} \]

\[ R = \]

<table>
<thead>
<tr>
<th>S</th>
<th>→</th>
<th>NP</th>
<th>VP</th>
</tr>
</thead>
<tbody>
<tr>
<td>VP</td>
<td>→</td>
<td>Vi</td>
<td></td>
</tr>
<tr>
<td>VP</td>
<td>→</td>
<td>Vt</td>
<td>NP</td>
</tr>
<tr>
<td>VP</td>
<td>→</td>
<td>VP</td>
<td>PP</td>
</tr>
<tr>
<td>NP</td>
<td>→</td>
<td>DT</td>
<td>NN</td>
</tr>
<tr>
<td>NP</td>
<td>→</td>
<td>NP</td>
<td>PP</td>
</tr>
<tr>
<td>PP</td>
<td>→</td>
<td>IN</td>
<td>NP</td>
</tr>
</tbody>
</table>

| Vi  | → | sleeps |
| Vt  | → | saw    |
| NN  | → | man    |
| NN  | → | woman  |
| NN  | → | telescope |
| DT  | → | the    |
| IN  | → | with   |
| IN  | → | in     |

Note: \( S = \text{sentence, VP = verb phrase, NP = noun phrase, PP = prepositional phrase, DT = determiner, Vi = intransitive verb, Vt = transitive verb, NN = noun, IN = preposition} \)
Left-Most Derivations

A left-most derivation is a sequence of strings $s_1 \ldots s_n$, where

- $s_1 = S$, the start symbol
- $s_n \in \Sigma^*$, i.e. $s_n$ is made up of terminal symbols only
- Each $s_i$ for $i = 2 \ldots n$ is derived from $s_{i-1}$ by picking the left-most non-terminal $X$ in $s_{i-1}$ and replacing it by some $\beta$ where $X \rightarrow \beta$ is a rule in $R$

For example: [S], [NP VP], [D N VP], [the N VP], [the man VP], [the man Vi], [the man sleeps]

Representation of a derivation as a tree:
Properties of CFGs

- A CFG defines a set of possible derivations
- A string \( s \in \Sigma^* \) is in the language defined by the CFG if there is at least one derivation that yields \( s \)
- Each string in the language generated by the CFG may have more than one derivation ("ambiguity")
A Fragment of a Noun Phrase Grammar

| NN   | ⇒  | box       |
| NN   | ⇒  | car       |
| NN   | ⇒  | mechanic  |
| NN   | ⇒  | pigeon    |
| DT   | ⇒  | the       |
| DT   | ⇒  | a         |

| JJ   | ⇒  | fast      |
| JJ   | ⇒  | metal     |
| JJ   | ⇒  | idealistic|
| JJ   | ⇒  | clay      |
Extended Grammar with Prepositional Phrases

\[
\begin{align*}
\tilde{\text{N}} & \Rightarrow \text{NN} \\
\tilde{\text{N}} & \Rightarrow \text{NN} \tilde{\text{N}} \\
\tilde{\text{N}} & \Rightarrow \text{JJ} \tilde{\text{N}} \\
\tilde{\text{N}} & \Rightarrow \tilde{\text{N}} \tilde{\text{N}} \\
\text{NP} & \Rightarrow \text{DT} \tilde{\text{N}}
\end{align*}
\]

<table>
<thead>
<tr>
<th>JJ</th>
<th>\Rightarrow \</th>
</tr>
</thead>
<tbody>
<tr>
<td>fast</td>
<td></td>
</tr>
<tr>
<td>metal</td>
<td></td>
</tr>
<tr>
<td>idealistic</td>
<td></td>
</tr>
<tr>
<td>clay</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JJ</th>
<th>\Rightarrow \</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td></td>
</tr>
<tr>
<td>under</td>
<td></td>
</tr>
<tr>
<td>of</td>
<td></td>
</tr>
<tr>
<td>on</td>
<td></td>
</tr>
<tr>
<td>with</td>
<td></td>
</tr>
<tr>
<td>as</td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{NN} & \Rightarrow \text{box} \\
\text{NN} & \Rightarrow \text{car} \\
\text{NN} & \Rightarrow \text{mechanic} \\
\text{NN} & \Rightarrow \text{pigeon} \\
\text{DT} & \Rightarrow \text{the} \\
\text{DT} & \Rightarrow \text{a}
\end{align*}
\]

Generates:

- in a box, under the box, the fast car mechanic under the pigeon in the box, …
Verbs, Verb Phrases and Sentences

- **Basic Verb Types**
  - $V_i =$ Intransitive verb  
    - e.g., sleeps, walks, laughs
  - $V_t =$ Transitive verb  
    - e.g., sees, saw, likes
  - $V_d =$ Ditransitive verb  
    - e.g., gave

- **Basic VP Rules**
  - $VP \rightarrow V_i$
  - $VP \rightarrow V_t \quad NP$
  - $VP \rightarrow V_d \quad NP \quad NP$

- **Basic S Rule**
  - $S \rightarrow NP \quad VP$

**Examples of VP:**
- sleeps, walks, likes
- the mechanic, gave
- the fast car

**Examples of S:**
- the man sleeps, the dog walks, the dog gave
- the mechanic the fast car
PPs Modifying Verb Phrases

A new rule: $\text{VP} \rightarrow \text{VP PP}$

New examples of VP:
sleeps in the car, walks like the mechanic, gave the mechanic the fast car on Tuesday, . . .
Complementizers and SBARs

- Complementizers
  \[ \text{COMP} = \text{complementizer} \quad \text{e.g., that} \]

- SBAR
  \[ \text{SBAR} \quad \rightarrow \quad \text{COMP} \quad \text{S} \]

Examples:
that the man sleeps, that the mechanic saw the dog . . .
More Verbs

- New Verb Types
  - V[5] e.g., said, reported
  - V[6] e.g., told, informed
  - V[7] e.g., bet

- New VP Rules
  - $VP \rightarrow V[5] \text{ SBAR}$
  - $VP \rightarrow V[6] \text{ NP SBAR}$
  - $VP \rightarrow V[7] \text{ NP NP SBAR}$

Examples of New VPs:
- said that the man sleeps
- told the dog that the mechanic likes the pigeon
- bet the pigeon $50 that the mechanic owns a fast car
Coordination

- A New Part-of-Speech:
  \[ \text{CC} = \text{Coordinator} \quad \text{e.g., and, or, but} \]

- New Rules
  \[
  \begin{align*}
  \text{NP} & \rightarrow \text{NP} \quad \text{CC} \quad \text{NP} \\
  \bar{\text{N}} & \rightarrow \bar{\text{N}} \quad \text{CC} \quad \bar{\text{N}} \\
  \text{VP} & \rightarrow \text{VP} \quad \text{CC} \quad \text{VP} \\
  \text{S} & \rightarrow \text{S} \quad \text{CC} \quad \text{S} \\
  \text{SBAR} & \rightarrow \text{SBAR} \quad \text{CC} \quad \text{SBAR}
  \end{align*}
  \]
Much more remains...

- **Agreement**
  
  The dogs laugh *vs.* The dog laughs

- **Wh-movement**
  
  The dog that the cat liked ___

- **Active vs. passive**
  
  The dog saw the cat *vs.*
  The cat was seen by the dog

- If you’re interested in reading more:

Attachment ambiguities

- A key parsing decision is how we ‘attach’ various constituents
  - PPs, adverbial or participial phrases, infinitives, coordinations, etc.

The board approved [its acquisition] [by Royal Trustco Ltd.] [of Toronto] [for $27 a share] [at its monthly meeting].
Attachment ambiguities

- A key parsing decision is how we ‘attach’ various constituents
  - PPs, adverbial or participial phrases, infinitives, coordinations, etc.

The board approved [its acquisition] [by Royal Trustco Ltd.]
[for $27 a share]
[at its monthly meeting].

- Catalan numbers: $C_n = \frac{(2n)!}{[(n+1)!n!]}$
- An exponentially growing series, which arises in many tree-like contexts:
  - E.g., the number of possible triangulations of a polygon with $n+2$ sides
    - Turns up in triangulation of probabilistic graphical models....
Attachments

- I cleaned the dishes from dinner
- I cleaned the dishes with detergent
- I cleaned the dishes in my pajamas
- I cleaned the dishes in the sink
Syntactic Ambiguities I

- Prepositional phrases:  
  *They cooked the beans in the pot on the stove with handles.*

- Particle vs. preposition:  
  *The lady dressed up the staircase.*

- Complement structures  
  *The tourists objected to the guide that they couldn’t hear.*
  *She knows you like the back of her hand.*

- Gerund vs. participial adjective  
  *Visiting relatives can be boring.*
  *Changing schedules frequently confused passengers.*
Syntactic Ambiguities II

• Modifier scope within NPs
  *impractical design requirements*
  *plastic cup holder*

• Multiple gap constructions
  *The chicken is ready to eat.*
  *The contractors are rich enough to sue.*

• Coordination scope:
  *Small rats and mice can squeeze into holes or cracks in the wall.*
Non-Local Phenomena

- **Dislocation / gapping**
  - Which book should Peter buy?
  - A debate arose which continued until the election.

- **Binding**
  - **Reference**
    - The IRS audits itself
  - **Control**
    - I want to go
    - I want you to go
Product Details (from Amazon)
Hardcover: 1779 pages
Publisher: Longman; 2nd Revised edition
Language: English
ISBN-10: 0582517346
Product Dimensions: 8.4 x 2.4 x 10 inches
Shipping Weight: 4.6 pounds
Parsing: Two problems to solve:
1. Repeated work...
Parsing: Two problems to solve:
1. Repeated work...
Parsing: Two problems to solve:
2. Choosing the correct parse

- How do we work out the correct attachment:
  - She saw the man with a telescope
  - Is the problem ‘AI complete’? Yes, but ...
  - Words are good predictors of attachment
    - Even absent full understanding
  
  - Moscow sent more than 100,000 soldiers into Afghanistan ...
  
  - Sydney Water breached an agreement with NSW Health ...

- Our statistical parsers will try to exploit such statistics.
Probabilistic Context Free Grammar
Probabilistic – or stochastic – context-free grammars (PCFGs)

• $G = (\Sigma, N, S, R, P)$
  • $T$ is a set of terminal symbols
  • $N$ is a set of nonterminal symbols
  • $S$ is the start symbol ($S \in N$)
  • $R$ is a set of rules/productions of the form $X \rightarrow \gamma$
  • $P$ is a probability function
    • $P: R \rightarrow [0,1]$
    • $\forall X \in N, \sum_{X \rightarrow \gamma \in R} P(X \rightarrow \gamma) = 1$

• A grammar $G$ generates a language model $L$.

$$P(T^*) = 1$$
## PCFG Example

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>NP</td>
<td>VP</td>
</tr>
<tr>
<td>VP</td>
<td>Vi</td>
<td></td>
</tr>
<tr>
<td>VP</td>
<td>Vt</td>
<td>NP</td>
</tr>
<tr>
<td>VP</td>
<td>VP</td>
<td>PP</td>
</tr>
<tr>
<td>NP</td>
<td>DT</td>
<td>NN</td>
</tr>
<tr>
<td>NP</td>
<td>NP</td>
<td>PP</td>
</tr>
<tr>
<td>PP</td>
<td>P</td>
<td>NP</td>
</tr>
</tbody>
</table>

- Probability of a tree $t$ with rules

$$ \alpha_1 \rightarrow \beta_1 \quad \alpha_2 \rightarrow \beta_2 \quad \cdots \quad \alpha_n \rightarrow \beta_n $$

is

$$ p(t) = \prod_{i=1}^{n} q(\alpha_i \rightarrow \beta_i) $$

where $q(\alpha \rightarrow \beta)$ is the probability for rule $\alpha \rightarrow \beta$.  

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Vi</td>
<td>sleeps</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>Vt</td>
<td>saw</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>NN</td>
<td>man</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>NN</td>
<td>woman</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>NN</td>
<td>telescope</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>DT</td>
<td>the</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>with</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>in</td>
<td>0.5</td>
<td></td>
</tr>
</tbody>
</table>
Example of a PCFG

<table>
<thead>
<tr>
<th>S</th>
<th>NP</th>
<th>VP</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>VP</td>
<td>Vi</td>
<td></td>
<td>0.4</td>
</tr>
<tr>
<td>VP</td>
<td>Vt</td>
<td>NP</td>
<td>0.4</td>
</tr>
<tr>
<td>VP</td>
<td>VP</td>
<td>PP</td>
<td>0.2</td>
</tr>
<tr>
<td>NP</td>
<td>DT</td>
<td>NN</td>
<td>0.3</td>
</tr>
<tr>
<td>NP</td>
<td>NP</td>
<td>PP</td>
<td>0.7</td>
</tr>
<tr>
<td>PP</td>
<td>P</td>
<td>NP</td>
<td>1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vi</th>
<th>sleeps</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vt</td>
<td>saw</td>
<td>1.0</td>
</tr>
<tr>
<td>NN</td>
<td>man</td>
<td>0.7</td>
</tr>
<tr>
<td>NN</td>
<td>woman</td>
<td>0.2</td>
</tr>
<tr>
<td>NN</td>
<td>telescope</td>
<td>0.1</td>
</tr>
<tr>
<td>DT</td>
<td>the</td>
<td>1.0</td>
</tr>
<tr>
<td>IN</td>
<td>with</td>
<td>0.5</td>
</tr>
<tr>
<td>IN</td>
<td>in</td>
<td>0.5</td>
</tr>
</tbody>
</table>

- Probability of a tree $t$ with rules

$$\alpha_1 \rightarrow \beta_1, \alpha_2 \rightarrow \beta_2, \ldots, \alpha_n \rightarrow \beta_n$$

is $p(t) = \prod_{i=1}^{n} q(\alpha_i \rightarrow \beta_i)$ where $q(\alpha \rightarrow \beta)$ is the probability for rule $\alpha \rightarrow \beta$. 
**Probability of a Parse**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>S ⇒ NP VP</td>
<td>1.0</td>
</tr>
<tr>
<td>VP ⇒ Vi</td>
<td>0.4</td>
</tr>
<tr>
<td>VP ⇒ Vt NP</td>
<td>0.4</td>
</tr>
<tr>
<td>VP ⇒ VP PP</td>
<td>0.2</td>
</tr>
<tr>
<td>NP ⇒ DT NN</td>
<td>0.3</td>
</tr>
<tr>
<td>NP ⇒ NP PP</td>
<td>0.7</td>
</tr>
<tr>
<td>PP ⇒ P NP</td>
<td>1.0</td>
</tr>
<tr>
<td>Vi ⇒ sleeps</td>
<td>1.0</td>
</tr>
<tr>
<td>Vt ⇒ saw</td>
<td>1.0</td>
</tr>
<tr>
<td>NN ⇒ man</td>
<td>0.7</td>
</tr>
<tr>
<td>NN ⇒ woman</td>
<td>0.2</td>
</tr>
<tr>
<td>NN ⇒ telescope</td>
<td>0.1</td>
</tr>
<tr>
<td>DT ⇒ the</td>
<td>1.0</td>
</tr>
<tr>
<td>IN ⇒ with</td>
<td>0.5</td>
</tr>
<tr>
<td>IN ⇒ in</td>
<td>0.5</td>
</tr>
</tbody>
</table>

**Example Parse Trees and Probabilities**

1. **Sentence**: The man sleeps
   - **Tree**: `S → NP VP` (1.0)
   - **Parse Tree**: `S → NP VP` (1.0)
   - **Probability**: `p(t_1) = 1.0 * 0.3 * 1.0 * 0.7 * 0.4 * 1.0`

2. **Sentence**: The man saw the woman with the telescope
   - **Tree**: `S → NP VP` (1.0)
   - **Parse Tree**: `S → NP VP` (1.0)
   - **Probability**: `p(t_2) = 1.8 * 0.3 * 1.0 * 0.7 * 0.2 * 0.4 * 1.0 * 0.3 * 1.0 * 0.2 * 0.4 * 0.5 * 0.3 * 1.0 * 0.1`
PCFGs: Learning and Inference

- **Model**
  - The probability of a tree $t$ with $n$ rules $\alpha_i \rightarrow \beta_i$, $i = 1..n$
  
  $$p(t) = \prod_{i=1}^{n} q(\alpha_i \rightarrow \beta_i)$$

- **Learning**
  - Read the rules off of labeled sentences, use ML estimates for probabilities
  
  $$q_{ML}(\alpha \rightarrow \beta) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)}$$
  
  - and use all of our standard smoothing tricks!

- **Inference**
  - For input sentence $s$, define $T(s)$ to be the set of trees whose *yield* is $s$ (whole leaves, read left to right, match the words in $s$)
  
  $$t^*(s) = \arg \max_{t \in T(s)} p(t)$$
Grammar Transforms
Chomsky Normal Form

- All rules are of the form $X \rightarrow Y Z$ or $X \rightarrow w$
  - $X, Y, Z \in N$ and $w \in \Sigma$
- A transformation to this form doesn’t change the weak generative capacity of a CFG
  - That is, it recognizes the same language
    - But maybe with different trees
- Empties and unaries are removed recursively
- n-ary rules are divided by introducing new nonterminals ($n > 2$)
A phrase structure grammar

S → NP VP
VP → V NP
VP → V NP PP
NP → NP NP
NP → NP PP
NP → N
NP → e
PP → P NP

N → people
N → fish
N → tanks
N → rods
V → people
V → fish
V → tanks
P → with
Chomsky Normal Form steps

S → NP VP
S → VP
VP → V NP
VP → V
VP → V NP PP
VP → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

N → people
N → fish
N → tanks
N → rods
V → people
V → fish
V → tanks
P → with
Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V
S → V
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
NP → N
PP → P NP
PP → P

N → people
N → fish
N → tanks
N → rods
V → people
V → fish
V → tanks
P → with
Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P
N → people
N → fish
N → tanks
N → rods
V → people
V → fish
V → tanks
S → people
S → fish
S → tanks
P → with
Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP
NP → NP PP
NP → PP
NP → N
PP → P NP
PP → P

N → people
N → fish
N → tanks
N → rods
V → people
S → people
VP → people
V → fish
S → fish
VP → fish
V → tanks
S → tanks
VP → tanks
P → with
Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP PP
NP → P NP
PP → P NP

NP → people
NP → fish
NP → tanks
NP → rods
V → people
S → people
VP → people
V → fish
S → fish
VP → fish
V → tanks
S → tanks
VP → tanks
P → with
PP → with
Chomsky Normal Form steps

\[
\begin{align*}
S & \rightarrow \text{NP VP} \\
\text{VP} & \rightarrow \text{V NP} \\
S & \rightarrow \text{V NP} \\
\text{VP} & \rightarrow \text{V @VP_V} \\
@\text{VP_V} & \rightarrow \text{NP PP} \\
S & \rightarrow \text{V @S_V} \\
@\text{S_V} & \rightarrow \text{NP PP} \\
\text{VP} & \rightarrow \text{V PP} \\
S & \rightarrow \text{V PP} \\
\text{NP} & \rightarrow \text{NP NP} \\
\text{NP} & \rightarrow \text{NP PP} \\
\text{NP} & \rightarrow \text{P NP} \\
\text{PP} & \rightarrow \text{P NP}
\end{align*}
\]

\[
\begin{align*}
\text{NP} & \rightarrow \text{people} \\
\text{NP} & \rightarrow \text{fish} \\
\text{NP} & \rightarrow \text{tanks} \\
\text{NP} & \rightarrow \text{rods} \\
\text{V} & \rightarrow \text{people} \\
\text{S} & \rightarrow \text{people} \\
\text{VP} & \rightarrow \text{people} \\
\text{V} & \rightarrow \text{fish} \\
\text{S} & \rightarrow \text{fish} \\
\text{VP} & \rightarrow \text{fish} \\
\text{V} & \rightarrow \text{tanks} \\
\text{S} & \rightarrow \text{tanks} \\
\text{VP} & \rightarrow \text{tanks} \\
\text{P} & \rightarrow \text{with} \\
\text{PP} & \rightarrow \text{with}
\end{align*}
\]
A phrase structure grammar

S → NP VP
VP → V NP
VP → V NP PP
NP → NP NP
NP → NP PP
NP → N
NP → e
PP → P NP

N → people
N → fish
N → tanks
N → rods
V → people
V → fish
V → tanks
P → with
Chomsky Normal Form steps

S → NP VP
VP → V NP
S → V NP
VP → V @VP_V
@VP_V → NP PP
S → V @S_V
@S_V → NP PP
VP → V PP
S → V PP
NP → NP NP
NP → NP PP
NP → P NP
PP → P NP

NP → people
NP → fish
NP → tanks
NP → rods
V → people
S → people
VP → people
V → fish
S → fish
VP → fish
V → tanks
S → tanks
VP → tanks
P → with
PP → with
Chomsky Normal Form

- You should think of this as a transformation for efficient parsing
- With some extra book-keeping in symbol names, you can even reconstruct the same trees with a detransform
- In practice full Chomsky Normal Form is a pain
  - Reconstructing n-aries is easy
  - Reconstructing unaries/empties is trickier

- **Binarization** is crucial for cubic time CFG parsing

- The rest isn’t necessary; it just makes the algorithms cleaner and a bit quicker
An example: before binarization...
After binarization...
Parsing
Constituency Parsing

PCFG

Rule Prob $\theta_i$

$S \rightarrow NP \ VP \quad \theta_0$

$NP \rightarrow NP \ NP \quad \theta_1$

$N \rightarrow \text{fish} \quad \theta_{42}$

$N \rightarrow \text{people} \quad \theta_{43}$

$V \rightarrow \text{fish} \quad \theta_{44}$

$\ldots$
Cocke-Kasami-Younger (CKY)
Constituency Parsing

fish  people  fish  tanks
Viterbi (Max) Scores

S \rightarrow \text{NP VP} \quad 0.9
S \rightarrow \text{VP} \quad 0.1
\text{VP} \rightarrow \text{V NP} \quad 0.5
\text{VP} \rightarrow \text{V} \quad 0.1
\text{VP} \rightarrow \text{V} \@\text{VP}_V \quad 0.3
\text{VP} \rightarrow \text{V} \ PP \quad 0.1
\@\text{VP}_V \rightarrow \text{NP PP} \quad 1.0
\text{NP} \rightarrow \text{NP NP} \quad 0.1
\text{NP} \rightarrow \text{NP} \ PP \quad 0.2
\text{NP} \rightarrow \text{N} \quad 0.7
\text{PP} \rightarrow \text{P} \ NP \quad 1.0
Extended CKY parsing

- Unaries can be incorporated into the algorithm
  - Messy, but doesn’t increase algorithmic complexity
- Empties can be incorporated
  - Use fenceposts
  - Doesn’t increase complexity; essentially like unaries

- Binarization is *vital*
  - Without binarization, you don’t get parsing cubic in the length of the sentence and in the number of nonterminals in the grammar
    - Binarization may be an explicit transformation or implicit in how the parser works (Early-style dotted rules), but it’s always there.
A Recursive Parser

\[ \text{bestScore}(X,i,j,s) \]

\[
\text{if} \ (j == i) \\
\quad \text{return} \ q(X->s[i])
\]

\[
\text{else} \\
\quad \text{return} \ \max_{k,X->YZ} q(X->YZ) \ * \\
\quad \text{bestScore}(Y,i,k,s) \ * \\
\quad \text{bestScore}(Z,k+1,j,s)
\]
The CKY algorithm (1960/1965)
... extended to unaries

function CKY(words, grammar) returns [most_probable_parse, prob]
  score = new double[#(words)+1][#(words)+1][#(nonterms)]
  back = new Pair[#(words)+1][#(words)+1][#(nonterms)]
  for i=0; i<#(words); i++
    for A in nonterms
      if A -> words[i] in grammar
        score[i][i+1][A] = P(A -> words[i])
    //handle unaries
    boolean added = true
    while added
      added = false
      for A, B in nonterms
        if score[i][i+1][B] > 0 && A->B in grammar
          prob = P(A->B)*score[i][i+1][B]
          if prob > score[i][i+1][A]
            score[i][i+1][A] = prob
            back[i][i+1][A] = B
            added = true
The CKY algorithm (1960/1965)  
... extended to unaries

for span = 2 to #(words)
    for begin = 0 to #(words) - span
        end = begin + span
        for split = begin + 1 to end - 1
            for A, B, C in nonterms
                prob = score[begin][split][B] * score[split][end][C] * P(A -> BC)
                if prob > score[begin][end][A]
                    score[begin][end][A] = prob
                    back[begin][end][A] = new Triple(split, B, C)
            //handle unaries
            boolean added = true
            while added
                added = false
                for A, B in nonterms
                    prob = P(A -> B) * score[begin][end][B];
                    if prob > score[begin][end][A]
                        score[begin][end][A] = prob
                        back[begin][end][A] = B
                        added = true
            return buildTree(score, back)
The grammar:
Binary, no epsilons,

<table>
<thead>
<tr>
<th>Rule</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>0.9</td>
</tr>
<tr>
<td>S → VP</td>
<td>0.1</td>
</tr>
<tr>
<td>VP → V NP</td>
<td>0.5</td>
</tr>
<tr>
<td>VP → V</td>
<td>0.1</td>
</tr>
<tr>
<td>VP → V @VP_V</td>
<td>0.3</td>
</tr>
<tr>
<td>VP → V PP</td>
<td>0.1</td>
</tr>
<tr>
<td>@VP_V → NP PP</td>
<td>1.0</td>
</tr>
<tr>
<td>NP → NP NP</td>
<td>0.1</td>
</tr>
<tr>
<td>NP → NP PP</td>
<td>0.2</td>
</tr>
<tr>
<td>NP → N</td>
<td>0.7</td>
</tr>
<tr>
<td>PP → P NP</td>
<td>1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>N → people</td>
<td>0.5</td>
</tr>
<tr>
<td>N → fish</td>
<td>0.2</td>
</tr>
<tr>
<td>N → tanks</td>
<td>0.2</td>
</tr>
<tr>
<td>N → rods</td>
<td>0.1</td>
</tr>
<tr>
<td>V → people</td>
<td>0.1</td>
</tr>
<tr>
<td>V → fish</td>
<td>0.6</td>
</tr>
<tr>
<td>V → tanks</td>
<td>0.3</td>
</tr>
<tr>
<td>P → with</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>fish</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

score[0][1] score[0][2] score[0][3] score[0][4]
score[1][2] score[1][3] score[1][4]
score[2][3] score[2][4]
score[3][4]
for i=0; i<#(words); i++
    for A in nonterms
        if A -> words[i] in grammar
            score[i][i+1][A] = P(A -> words[i]);
### Grammar Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>0.9</td>
</tr>
<tr>
<td>S → VP</td>
<td>0.1</td>
</tr>
<tr>
<td>VP → V NP</td>
<td>0.5</td>
</tr>
<tr>
<td>VP → V</td>
<td>0.1</td>
</tr>
<tr>
<td>VP → V @VP_V</td>
<td>0.3</td>
</tr>
<tr>
<td>VP → V PP</td>
<td>0.1</td>
</tr>
<tr>
<td>@VP_V → NP PP</td>
<td>1.0</td>
</tr>
<tr>
<td>NP → NP NP</td>
<td>0.1</td>
</tr>
<tr>
<td>NP → NP PP</td>
<td>0.2</td>
</tr>
<tr>
<td>NP → N</td>
<td>0.7</td>
</tr>
<tr>
<td>PP → P NP</td>
<td>1.0</td>
</tr>
<tr>
<td>N → people</td>
<td>0.5</td>
</tr>
<tr>
<td>N → fish</td>
<td>0.2</td>
</tr>
<tr>
<td>N → tanks</td>
<td>0.2</td>
</tr>
<tr>
<td>N → rods</td>
<td>0.1</td>
</tr>
<tr>
<td>V → people</td>
<td>0.1</td>
</tr>
<tr>
<td>V → fish</td>
<td>0.6</td>
</tr>
<tr>
<td>V → tanks</td>
<td>0.3</td>
</tr>
<tr>
<td>P → with</td>
<td>1.0</td>
</tr>
</tbody>
</table>

### Chart

```
<table>
<thead>
<tr>
<th></th>
<th>fish</th>
<th>people</th>
<th>fish</th>
<th>tanks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N → fish 0.2</td>
<td>V → fish 0.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>N → people 0.5</td>
<td>V → people 0.1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>N → fish 0.2</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

### Additional Code

```java
// handle unaries
boolean added = true
while added
  added = false
  for A, B in nonterms
    if score[i][i+1][B] > 0 && A>B in grammar
      prob = P(A>B)*score[i][i+1][B]
      if(prob > score[i][i+1][A])
        score[i][i+1][A] = prob
        back[i][i+1][A] = B
        added = true
```
| S → NP VP | 0.9 |
| S → VP | 0.1 |
| VP → V NP | 0.5 |
| VP → V | 0.1 |
| VP → V @VP_V | 0.3 |
| VP → V PP | 0.1 |
| @VP_V → NP PP | 1.0 |
| NP → NP NP | 0.1 |
| NP → NP PP | 0.2 |
| NP → N | 0.7 |
| PP → P NP | 1.0 |
| N → people | 0.5 |
| N → fish | 0.2 |
| N → tanks | 0.2 |
| N → rods | 0.1 |
| V → people | 0.1 |
| V → fish | 0.6 |
| V → tanks | 0.3 |
| P → with | 1.0 |

The diagram shows the probability distribution for different sentences, with the following rules:

- **S → NP VP**: 0.9
- **S → VP**: 0.1
- **VP → V NP**: 0.5
- **VP → V**: 0.1
- **VP → V @VP_V**: 0.3
- **VP → V PP**: 0.1
- **@VP_V → NP PP**: 1.0
- **NP → NP NP**: 0.1
- **NP → NP PP**: 0.2
- **NP → N**: 0.7
- **PP → P NP**: 1.0
- **N → people**: 0.5
- **N → fish**: 0.2
- **N → tanks**: 0.2
- **N → rods**: 0.1
- **V → people**: 0.1
- **V → fish**: 0.6
- **V → tanks**: 0.3
- **P → with**: 1.0

The diagram is color-coded to represent different probabilities:
- **Fish**: Blue
- **People**: Orange
- **Tanks**: Yellow
- **Rods**: Green

The underlying logic includes probability evaluations and backtracking mechanisms, as indicated by the code snippets and conditional logic within the diagram.
//handle unaries

boolean added = true
while added
  added = false
  for A, B in nonterms
    prob = P(A->B)*score[begin][end][B];
    if prob > score[begin][end][A]
      score[begin][end][A] = prob
      back[begin][end][A] = B
      added = true
<table>
<thead>
<tr>
<th>Nonterm</th>
<th>Prog</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>0.9</td>
<td></td>
</tr>
<tr>
<td>S → VP</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>VP → V NP</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>VP → V</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>VP → V @VP_V</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>@VP_V → NP PP</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>NP → NP NP</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>NP → NP PP</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>NP → N</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>PP → P NP</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>N → people</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>N → fish</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>N → tanks</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>N → rods</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>V → people</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>V → fish</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>V → tanks</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>P → with</td>
<td>1.0</td>
<td></td>
</tr>
</tbody>
</table>

**Recursion Procedure**

For split = begin + 1 to end - 1
For A, B, C in nonterms
    prob = score[begin][split][B] * score[split][end][C] * P(A -> BC)
    if prob > score[begin][end][A]
        score[begin][end][A] = prob
        back[begin][end][A] = new Triple(split, B, C)

**Production Rules**

- **S → NP VP** 0.9
- **S → VP** 0.1
- **NP → NP NP** 0.0049
- **NP → N 0.14**
- **VP → V NP** 0.105
- **NP → V 0.06**
- **S → NP VP** 0.006
- **N → fish 0.2**
- **V → fish 0.6**
- **NP → N 0.14**
- **VP → V 0.06**
- **S → VP 0.006**
- **N → people 0.5**
- **V → people 0.1**
- **NP → N 0.35**
- **VP → V 0.01**
- **S → NP VP** 0.0189
- **NP → NP NP** 0.0049
- **VP → V NP** 0.007
- **S → VP 0.0105**
- **N → fish 0.2**
- **V → fish 0.6**
- **NP → N 0.14**
- **VP → V 0.06**
- **S → VP 0.001**
- **N → tanks 0.2**
- **V → tanks 0.1**
- **NP → N 0.14**
- **VP → V 0.03**
- **S → VP 0.003**
for split = begin+1 to end-1
for A,B,C in nonterms
    prob = score[begin][split][B]*score[split][end][C]*P(A->BC)
    if prob > score[begin][end][A]
        score[begin][end][A] = prob
        back[begin][end][A] = new Triple(split,B,C)
<table>
<thead>
<tr>
<th></th>
<th>fish</th>
<th>people</th>
<th>fish</th>
<th>tanks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N → fish 0.2</td>
<td>NP → NP NP 0.0049</td>
<td>NP → NP NP 0.0000686</td>
<td>NP → NP NP 0.0000686</td>
</tr>
<tr>
<td></td>
<td>V → fish 0.6</td>
<td>VP → V NP 0.105</td>
<td>VP → V NP 0.00147</td>
<td>VP → V NP 0.000098</td>
</tr>
<tr>
<td></td>
<td>NP → N 0.14</td>
<td>S → VP 0.0105</td>
<td>S → VP 0.00882</td>
<td>S → VP 0.01323</td>
</tr>
<tr>
<td>1</td>
<td>N → people 0.5</td>
<td>NP → NP NP 0.0049</td>
<td>NP → NP NP 0.0000686</td>
<td>NP → NP NP 0.000196</td>
</tr>
<tr>
<td></td>
<td>V → people 0.1</td>
<td>VP → V NP 0.007</td>
<td>VP → V NP 0.000098</td>
<td>VP → V NP 0.00196</td>
</tr>
<tr>
<td></td>
<td>NP → N 0.35</td>
<td>S → NP VP 0.0189</td>
<td>S → NP VP 0.001323</td>
<td>S → NP VP 0.00142</td>
</tr>
<tr>
<td>2</td>
<td>N → fish 0.2</td>
<td>VP → V NP 0.01</td>
<td>VP → V NP 0.004</td>
<td></td>
</tr>
<tr>
<td></td>
<td>V → fish 0.6</td>
<td>S → VP 0.006</td>
<td>S → VP 0.006</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NP → N 0.14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>N → tanks 0.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>V → tanks 0.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NP → N 0.14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>VP → V 0.03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S → VP 0.003</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>P → with 1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Call buildTree(score, back) to get the best parse
Evaluating constituency parsing

Gold standard brackets:  \( S-(0:11), \text{NP-(0:2)}, \text{VP-(2:9)}, \text{VP-(3:9)}, \text{NP-(4:6)}, \text{PP-(6:9)}, \text{NP-(7,9)}, \text{NP-(9:10)} \)

```
S
  / \NP
  |   /VP
  |   / VP
  |   / VBD
  |   / NNS
  |   /Sales
  |   /exectives
  |   / were
  |   / VBG
  |   / examining
  |   / DT
  |   / NNS
  |   / the
  |   / IN
  |   / NNS
  |   / figures
  |   / IN
  |   / NNS
  |   / with
  |   / JJ
  |   / NN
  |   / great
  |   / NP
  |   / Care
  |   / yesterday
```

Candidate brackets:  \( S-(0:11), \text{NP-(0:2)}, \text{VP-(2:10)}, \text{VP-(3:10)}, \text{NP-(4:6)}, \text{PP-(6-10)}, \text{NP-(7,10)} \)

```
S
  / \NP
  |   /VP
  |   / VP
  |   / VBD
  |   / NNS
  |   /Sales
  |   / executives
  |   / were
  |   / VBG
  |   / examining
  |   / DT
  |   / NNS
  |   / the
  |   / IN
  |   / NNS
  |   / figures
  |   / IN
  |   / NNS
  |   / with
  |   / JJ
  |   / NN
  |   / great
  |   / NP
  |   / Care
  |   / yesterday
```
Evaluating constituency parsing

Gold standard brackets:
S-(0:11), NP-(0:2), VP-(2:9), VP-(3:9), NP-(4:6), PP-(6-9), NP-(7,9), NP-(9:10)

Candidate brackets:
S-(0:11), NP-(0:2), VP-(2:10), VP-(3:10), NP-(4:6), PP-(6-10), NP-(7,10)

Labeled Precision 3/7 = 42.9%
Labeled Recall 3/8 = 37.5%
LP/LR F1 40.0%
Tagging Accuracy 11/11 = 100.0%
How good are PCFGs?

- Penn WSJ parsing accuracy: about 73.7% LP/LR F1
- Robust
  - Usually admit everything, but with low probability
- Partial solution for grammar ambiguity
  - A PCFG gives some idea of the plausibility of a parse
  - But not so good because the independence assumptions are too strong
- Give a probabilistic language model
  - But in the simple case it performs worse than a trigram model
- The problem seems to be that PCFGs lack the lexicalization of a trigram model
Weaknesses of PCFGs
Weaknesses

- Lack of sensitivity to structural frequencies
- Lack of sensitivity to lexical information
- (A word is independent of the rest of the tree given its POS!)
A Case of PP Attachment Ambiguity

(a)  
S  
  NP  
    NNS  
      workers  
    VP  
      VBD  
        dumped  
      NP  
        NNS  
          sacks  
      IN  
        into  
      NP  
        DT  
          a  
        NN  
          bin  

(b)  
S  
  NP  
    NNS  
      workers  
    VP  
      VBD  
        dumped  
      NP  
        NNS  
          sacks  
      IN  
        into  
      NP  
        DT  
          a  
        NN  
          bin
<table>
<thead>
<tr>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NP</strong> → NNS</td>
</tr>
<tr>
<td><strong>VP</strong> → <strong>VP</strong> <strong>PP</strong></td>
</tr>
<tr>
<td>VP → VBD NP</td>
</tr>
<tr>
<td>NP → NNS</td>
</tr>
<tr>
<td>PP → IN NP</td>
</tr>
<tr>
<td>NP → DT NN</td>
</tr>
<tr>
<td>NNS → workers</td>
</tr>
<tr>
<td>VBD → dumped</td>
</tr>
<tr>
<td>NNS → sacks</td>
</tr>
<tr>
<td>IN → into</td>
</tr>
<tr>
<td>DT → a</td>
</tr>
<tr>
<td>NN → bin</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NP</strong> → <strong>NP</strong> <strong>PP</strong></td>
</tr>
<tr>
<td>VP → VBD NP</td>
</tr>
<tr>
<td>NP → NNS</td>
</tr>
<tr>
<td>PP → IN NP</td>
</tr>
<tr>
<td>NP → DT NN</td>
</tr>
<tr>
<td>NNS → workers</td>
</tr>
<tr>
<td>VBD → dumped</td>
</tr>
<tr>
<td>NNS → sacks</td>
</tr>
<tr>
<td>IN → into</td>
</tr>
<tr>
<td>DT → a</td>
</tr>
<tr>
<td>NN → bin</td>
</tr>
</tbody>
</table>

If $q(NP \rightarrow NP \ PP) > q(VP \rightarrow VP \ PP)$ then (b) is more probable, else (a) is more probable.

**Attachment decision is completely independent of the words**
A Case of Coordination Ambiguity
Here the two parses have identical rules, and therefore have identical probability under any assignment of PCFG rule probabilities.
Structural Preferences: Close Attachment

Example: president of a company in Africa

Both parses have the same rules, therefore receive same probability under a PCFG

“Close attachment” (structure (a)) is twice as likely in Wall Street Journal text.
Structural Preferences: Close Attachment

- Example: John was believed to have been shot by Bill

- Low attachment analysis (Bill does the shooting) contains same rules as high attachment analysis (Bill does the believing)
  - Two analyses receive the same probability
PCFGs and Independence

The symbols in a PCFG define independence assumptions:

- At any node, the material inside that node is independent of the material outside that node, given the label of that node.
- Any information that statistically connects behavior inside and outside a node must flow through that node’s label.

```
S  →  NP  VP
NP →  DT  NN
```

NP → DT NN
VP → NN VP
S → NP VP

Non-Independence I

- The independence assumptions of a PCFG are often too strong

Example: the expansion of an NP is highly dependent on the parent of the NP (i.e., subjects vs. objects)
Non-Independence II

- Symptoms of overly strong assumptions:
  - Rewrites get used where they don’t belong

In the PTB, this construction is for possessives
Refining the Grammar Symbols

- We can relax independence assumptions by encoding dependencies into the PCFG symbols, by state splitting:

  Parent annotation
  [Johnson 98]

  Marking possessive NPs

- Too much state-splitting \(\rightarrow\) sparseness (no smoothing used!)
- What are the most useful features to encode?
Linguistics in Unlexicalized Parsing
Horizontal Markovization

- Horizontal Markovization: Merges States
Vertical Markovization

- Vertical Markov order: rewrites depend on past $k$ ancestor nodes. (i.e., parent annotation)

![Diagram showing Vertical Markovization](image)

<table>
<thead>
<tr>
<th>Model</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>v=h=2v</td>
<td>77.8</td>
<td>7.5K</td>
</tr>
</tbody>
</table>
Unary Splits

- Problem: unary rewrites are used to transmute categories so a high-probability rule can be used.

Solution: Mark unary rewrite sites with -U

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>77.8</td>
<td>7.5K</td>
</tr>
<tr>
<td>UNARY</td>
<td>78.3</td>
<td>8.0K</td>
</tr>
</tbody>
</table>
Tag Splits

- Problem: Treebank tags are too coarse.

- Example: SBAR sentential complementizers (*that, whether, if*), subordinating conjunctions (*while, after*), and true prepositions (*in, of, to*) are all tagged IN.

- Partial Solution:
  - Subdivide the IN tag.

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>78.3</td>
<td>8.0K</td>
</tr>
<tr>
<td>SPLIT-IN</td>
<td>80.3</td>
<td>8.1K</td>
</tr>
</tbody>
</table>
Other Tag Splits

- **UNARY-DT**: mark demonstratives as DT\(^\text{U}\) (“the X” vs. “those”)
- **UNARY-RB**: mark phrasal adverbs as RB\(^\text{U}\) (“quickly” vs. “very”)
- **TAG-PA**: mark tags with non-canonical parents (“not” is an RB\(^\text{VP}\) )
- **SPLIT-AUX**: mark auxiliary verbs with –AUX [cf. Charniak 97]
- **SPLIT-CC**: separate “but” and “&” from other conjunctions
- **SPLIT-%**: “%” gets its own tag.

<table>
<thead>
<tr>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>80.4</td>
<td>8.1K</td>
</tr>
<tr>
<td>80.5</td>
<td>8.1K</td>
</tr>
<tr>
<td>81.2</td>
<td>8.5K</td>
</tr>
<tr>
<td>81.6</td>
<td>9.0K</td>
</tr>
<tr>
<td>81.7</td>
<td>9.1K</td>
</tr>
<tr>
<td>81.8</td>
<td>9.3K</td>
</tr>
</tbody>
</table>
Yield Splits

- Problem: sometimes the behavior of a category depends on something inside its future yield.

- Examples:
  - Possessive NPs
  - Finite vs. infinite VPs
  - Lexical heads!

- Solution: annotate future elements into nodes.

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag splits</td>
<td>82.3</td>
<td>9.7K</td>
</tr>
<tr>
<td>POSS-NP</td>
<td>83.1</td>
<td>9.8K</td>
</tr>
<tr>
<td>SPLIT-VP</td>
<td>85.7</td>
<td>10.5K</td>
</tr>
</tbody>
</table>
Distance / Recursion Splits

- Problem: vanilla PCFGs cannot distinguish attachment heights.

- Solution: mark a property of higher or lower sites:
  - Contains a verb.
  - Is (non)-recursive.
    - Base NPs [cf. Collins 99]
    - Right-recursive NPs

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>85.7</td>
<td>10.5K</td>
</tr>
<tr>
<td>BASE-NP</td>
<td>86.0</td>
<td>11.7K</td>
</tr>
<tr>
<td>DOMINATES-V</td>
<td>86.9</td>
<td>14.1K</td>
</tr>
<tr>
<td>RIGHT-REC-NP</td>
<td>87.0</td>
<td>15.2K</td>
</tr>
</tbody>
</table>
A Fully Annotated Tree

```
ROOT
  |__ S^ROOT-v
  |
  |__ "S
  |   |
  |   |__ NP^S-B
  |       |
  |       |__ DT-U^NP
  |           |
  |           |__ "This"
  |             |
  |             |__ "is"
  |               |
  |               |__ NP^VP-B
  |                   |
  |                   |__ "NN^NP" "NN^NP"
  |                       |
  |                       |__ "panic" "buying"
  |                         |
  |                         |__ "^S"
  |                           |
  |                           |__ "^S"
```
Final Test Set Results

<table>
<thead>
<tr>
<th>Parser</th>
<th>LP</th>
<th>LR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Magerman 95</td>
<td>84.9</td>
<td>84.6</td>
<td>84.7</td>
</tr>
<tr>
<td>Collins 96</td>
<td>86.3</td>
<td>85.8</td>
<td>86.0</td>
</tr>
<tr>
<td><strong>Klein &amp; Manning 03</strong></td>
<td><strong>86.9</strong></td>
<td><strong>85.7</strong></td>
<td><strong>86.3</strong></td>
</tr>
<tr>
<td>Charniak 97</td>
<td>87.4</td>
<td>87.5</td>
<td>87.4</td>
</tr>
<tr>
<td>Collins 99</td>
<td>88.7</td>
<td>88.6</td>
<td>88.6</td>
</tr>
</tbody>
</table>

- Beats “first generation” lexicalized parsers
Lexicalised PCFGs
Heads in Context Free Rules

Add annotations specifying the “head” of each rule:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Head</th>
<th>Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>NP</td>
<td>VP</td>
</tr>
<tr>
<td>VP</td>
<td>Vi</td>
<td></td>
</tr>
<tr>
<td>VP</td>
<td>Vt</td>
<td>NP</td>
</tr>
<tr>
<td>VP</td>
<td>VP</td>
<td>PP</td>
</tr>
<tr>
<td>NP</td>
<td>DT</td>
<td>NN</td>
</tr>
<tr>
<td>NP</td>
<td>NP</td>
<td>PP</td>
</tr>
<tr>
<td>PP</td>
<td>IN</td>
<td>NP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Head</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vi</td>
<td>sleeps</td>
</tr>
<tr>
<td>Vt</td>
<td>saw</td>
</tr>
<tr>
<td>NN</td>
<td>man</td>
</tr>
<tr>
<td>NN</td>
<td>woman</td>
</tr>
<tr>
<td>NN</td>
<td>telescope</td>
</tr>
<tr>
<td>DT</td>
<td>the</td>
</tr>
<tr>
<td>IN</td>
<td>with</td>
</tr>
<tr>
<td>IN</td>
<td>in</td>
</tr>
</tbody>
</table>
Heads

- Each context-free rule has one “special” child that is the head of the rule. e.g.,
  
  \[
  S \Rightarrow NP \; VP \quad (VP \; is \; the \; head)
  \]
  
  \[
  VP \Rightarrow Vt \; NP \quad (Vt \; is \; the \; head)
  \]
  
  \[
  NP \Rightarrow DT \; NN \; NN \quad (NN \; is \; the \; head)
  \]

- A core idea in syntax
  (e.g., see X-bar Theory, Head-Driven Phrase Structure Grammar)

- Some intuitions:
  
  - The central sub-constituent of each rule.
  - The semantic predicate in each rule.
Rules to Recover Heads: An Example for NPs

If the rule contains NN, NNS, or NNP:
Choose the rightmost NN, NNS, or NNP

Else If the rule contains an NP: Choose the leftmost NP

Else If the rule contains a JJ: Choose the rightmost JJ

Else If the rule contains a CD: Choose the rightmost CD

Else Choose the rightmost child

e.g.,

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>⇒</td>
<td>DT</td>
<td>NNP</td>
<td>NN</td>
</tr>
<tr>
<td>NP</td>
<td>⇒</td>
<td>DT</td>
<td>NN</td>
<td>NNP</td>
</tr>
<tr>
<td>NP</td>
<td>⇒</td>
<td>NP</td>
<td>PP</td>
<td></td>
</tr>
<tr>
<td>NP</td>
<td>⇒</td>
<td>DT</td>
<td>JJ</td>
<td></td>
</tr>
<tr>
<td>NP</td>
<td>⇒</td>
<td>DT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Rules to Recover Heads: An Example for VPs

If the rule contains Vi or Vt: Choose the leftmost Vi or Vt

Else If the rule contains an VP: Choose the leftmost VP

Else Choose the leftmost child

e.g.,

\[
\begin{align*}
\text{VP} & \Rightarrow \text{Vt} \quad \text{NP} \\
\text{VP} & \Rightarrow \text{VP} \quad \text{PP}
\end{align*}
\]
Adding Headwords to Trees
Adding Headwords to Trees

```
S
  ├── NP
  │   └── DT
  │       └── the
  └── VP
      └── Vt
          └── NP
              └── DT
                  └── the
          └── NN
              └── witness

S(questioned)
  ├── NP
  │   └── DT
  │       └── the
  │       └── NN
  │           └── lawyer
  └── VP
      └── Vt
          └── NP
              └── DT
                  └── the
              └── NN
                  └── witness
```
Adding Headwords to Trees

- A constituent receives its headword from its head child.

```
S  =>  NP   VP  (S receives headword from VP)
VP =>  Vt   NP  (VP receives headword from Vt)
NP =>  DT   NN  (NP receives headword from NN)
```
Lexicalized CFGs in Chomsky Normal Form

- $N$ is a set of non-terminal symbols
- $\Sigma$ is a set of terminal symbols
- $R$ is a set of rules which take one of three forms:
  - $X(h) \rightarrow_1 Y_1(h) \ Y_2(w)$ for $X \in N$, and $Y_1, Y_2 \in N$, and $h, w \in \Sigma$
  - $X(h) \rightarrow_2 Y_1(w) \ Y_2(h)$ for $X \in N$, and $Y_1, Y_2 \in N$, and $h, w \in \Sigma$
  - $X(h) \rightarrow h$ for $X \in N$, and $h \in \Sigma$
- $S \in N$ is a distinguished start symbol
Example

\[\begin{align*}
S(\text{saw}) & \rightarrow_2 \text{NP(}\text{man}\text{)} & \text{VP(}\text{saw}\text{)} \\
\text{VP(}\text{saw}\text{)} & \rightarrow_1 \text{Vt(}\text{saw}\text{)} & \text{NP(}\text{dog}\text{)} \\
\text{NP(}\text{man}\text{)} & \rightarrow_2 \text{DT(}\text{the}\text{)} & \text{NN(}\text{man}\text{)} \\
\text{NP(}\text{dog}\text{)} & \rightarrow_2 \text{DT(}\text{the}\text{)} & \text{NN(}\text{dog}\text{)} \\
\text{Vt(}\text{saw}\text{)} & \rightarrow & \text{saw} \\
\text{DT(}\text{the}\text{)} & \rightarrow & \text{the} \\
\text{NN(}\text{man}\text{)} & \rightarrow & \text{man} \\
\text{NN(}\text{dog}\text{)} & \rightarrow & \text{dog}
\end{align*}\]
Lexicalized CKY

bestScore(X, i, j, h)
if (j = i)
    return score(X, s[i])
else
    return max_{k, h, X \rightarrow YZ} max_{k, h, X \rightarrow YZ} score(X[h] \rightarrow Y[h]Z[w]) * bestScore(Y, i, k, h) * bestScore(Z, k, j, w)

score(X[h] \rightarrow Y[w]Z[h]) * bestScore(Y, i, k, w) * bestScore(Z, k, j, h)
Parsing with Lexicalized CFGs

The new form of grammar looks just like a Chomsky normal form CFG, but with potentially \( O(|\Sigma|^2 \times |N|^3) \) possible rules.

Naively, parsing an \( n \) word sentence using the dynamic programming algorithm will take \( O(n^3|\Sigma|^2|N|^3) \) time. But \(|\Sigma|\) can be huge!!

Crucial observation: at most \( O(n^2 \times |N|^3) \) rules can be applicable to a given sentence \( w_1, w_2, \ldots w_n \) of length \( n \). This is because any rules which contain a lexical item that is not one of \( w_1 \ldots w_n \), can be safely discarded.

The result: we can parse in \( O(n^5|N|^3) \) time.
Pruning with Beams

- The Collins parser prunes with per-cell beams [Collins 99]
  - Essentially, run the $O(n^5)$ CKY
  - Remember only a few hypotheses for each span $<i,j>$.
  - If we keep $K$ hypotheses at each span, then we do at most $O(nK^2)$ work per span (why?)
  - Keeps things more or less cubic

- Also: certain spans are forbidden entirely on the basis of punctuation (crucial for speed)
Parameter Estimation

\[ p(t) = q(S(\text{saw}) \rightarrow_2 NP(\text{man}) \ VP(\text{saw})) \]
\[ \times q(NP(\text{man}) \rightarrow_2 DT(\text{the}) \ NN(\text{man})) \]
\[ \times q(VP(\text{saw}) \rightarrow_1 VP(\text{saw}) \ PP(\text{with})) \]
\[ \times q(VP(\text{saw}) \rightarrow_1 Vt(\text{saw}) \ NP(\text{dog})) \]
\[ \times q(PP(\text{with}) \rightarrow_1 IN(\text{with}) \ NP(\text{telescope})) \]
\[ \times \ldots \]
A Model from Charniak (1997)

- An example parameter in a Lexicalized PCFG:

\[ q(S(\text{saw}) \rightarrow_2 \text{NP(\text{man}) VP(\text{saw})}) \]

- First step: decompose this parameter into a product of two parameters

\[ q(S(\text{saw}) \rightarrow_2 \text{NP(\text{man}) VP(\text{saw})}) = q(S \rightarrow_2 \text{NP VP}|S, \text{saw}) \times q(\text{man}|S \rightarrow_2 \text{NP VP, saw}) \]
A Model from Charniak (1997)

\[ q(S(saw) \rightarrow_2 NP(man) \text{ VP}(saw)) = q(S \rightarrow_2 NP \text{ VP}|S, \text{ saw}) \times q(man|S \rightarrow_2 NP \text{ VP}, \text{ saw}) \]

- **Second step:** use smoothed estimation for the two parameter estimates

\[ q(S \rightarrow_2 NP \text{ VP}|S, \text{ saw}) = \lambda_1 \times q_{ML}(S \rightarrow_2 NP \text{ VP}|S, \text{ saw}) + \lambda_2 \times q_{ML}(S \rightarrow_2 NP \text{ VP}|S) \]

\[ q(man|S \rightarrow_2 NP \text{ VP}, \text{ saw}) = \lambda_3 \times q_{ML}(man|S \rightarrow_2 NP \text{ VP}, \text{ saw}) + \lambda_4 \times q_{ML}(man|S \rightarrow_2 NP \text{ VP}) + \lambda_5 \times q_{ML}(man|NP) \]
Other Details

- Need to deal with rules with more than two children, e.g.,
  \[ VP(told) \rightarrow V(told) \ NP(him) \ PP(on) \ SBAR(that) \]

- Need to incorporate parts of speech (useful in smoothing)
  \[ VP-V(told) \rightarrow V(told) \ NP-PRP(him) \ PP-IN(on) \ SBAR-COMP(that) \]

- Need to encode preferences for close attachment
  John was believed to have been shot by Bill
# Final Test Set Results

<table>
<thead>
<tr>
<th>Parser</th>
<th>LP</th>
<th>LR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Magerman 95</td>
<td>84.9</td>
<td>84.6</td>
<td>84.7</td>
</tr>
<tr>
<td>Collins 96</td>
<td>86.3</td>
<td>85.8</td>
<td>86.0</td>
</tr>
<tr>
<td>Klein &amp; Manning 03</td>
<td>86.9</td>
<td>85.7</td>
<td>86.3</td>
</tr>
<tr>
<td>Charniak 97</td>
<td>87.4</td>
<td>87.5</td>
<td>87.4</td>
</tr>
<tr>
<td>Collins 99</td>
<td>88.7</td>
<td>88.6</td>
<td>88.6</td>
</tr>
</tbody>
</table>
Analysis/Evaluation (Method 2)

\[
\begin{align*}
S & \rightarrow_{2} NP \ VP \\
NP & \rightarrow_{2} DT \ NN \\
VP & \rightarrow_{1} VP \ PP \\
VP & \rightarrow_{1} Vt \ NP \\
NP & \rightarrow_{2} DT \ NN \\
PP & \rightarrow_{1} IN \ NP \\
NP & \rightarrow_{2} DT \ NN
\end{align*}
\]
Dependency Accuracies

- All parses for a sentence with $n$ words have $n$ dependencies. 
  *Report a single figure, dependency accuracy*

- Results from Collins, 2003: 88.3% dependency accuracy

- Can calculate precision/recall on particular dependency types.
  - e.g., look at all subject/verb dependencies $\Rightarrow$
    all dependencies with label $S \rightarrow_2$ NP VP

Recall = $\frac{\text{number of subject/verb dependencies correct}}{\text{number of subject/verb dependencies in gold standard}}$

Precision = $\frac{\text{number of subject/verb dependencies correct}}{\text{number of subject/verb dependencies in parser's output}}$
Strengths and Weaknesses of Modern Parsers

(Numbers taken from Collins (2003))

- Subject-verb pairs: over 95% recall and precision
- Object-verb pairs: over 92% recall and precision
- Other arguments to verbs: \( \approx 93\% \) recall and precision
- Non-recursive NP boundaries: \( \approx 93\% \) recall and precision
- PP attachments: \( \approx 82\% \) recall and precision
- Coordination ambiguities: \( \approx 61\% \) recall and precision
Modern Parsers
The Game of Designing a Grammar

- Annotation refines base treebank symbols to improve statistical fit of the grammar
  - Parent annotation [Johnson ’98]
  - Head lexicalization [Collins ’99, Charniak ’00]
  - Automatic clustering?
Manual Splits

- Manually split categories
  - NP: subject vs object
  - DT: determiners vs demonstratives
  - IN: sentential vs prepositional

- Advantages:
  - Fairly compact grammar
  - Linguistic motivations

- Disadvantages:
  - Performance leveled out
  - Manually annotated
Learning Latent Annotations

Latent Annotations:

- Brackets are known
- Base categories are known
- Hidden variables for subcategories

He was right.

Can learn with EM: like Forward-Backward for HMMs.
Automatic Annotation Induction

- Advantages:
  - Automatically learned:
    - Label \textit{all} nodes with latent variables.
    - Same number $k$ of subcategories for all categories.

- Disadvantages:
  - Grammar gets too large
  - Most categories are oversplit while others are undersplit.

<table>
<thead>
<tr>
<th>Model</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Klein &amp; Manning ’03</td>
<td>86.3</td>
</tr>
<tr>
<td>Matsuzaki et al. ’05</td>
<td>86.7</td>
</tr>
</tbody>
</table>
Refinement of the DT tag

DT

- the (0.50)
- a (0.24)
- The (0.08)

- a (0.61)
- the (0.19)
- an (0.11)

- the (0.80)
- The (0.15)
- a (0.01)

- this (0.39)
- that (0.28)
- That (0.11)

- some (0.20)
- all (0.19)
- those (0.12)
Hierarchical refinement

- Repeatedly learn more fine-grained subcategories
- Start with two (per non-terminal), then keep splitting
- Initialize each EM run with the output of the last
Adaptive Splitting

- Want to split complex categories more
- **Idea:** split everything, roll back splits which were least useful

[Diagram of adaptive splitting with probabilities]

[Petrov et al. 06]
Adaptive Splitting

- Evaluate loss in likelihood from removing each split = \[ \frac{Data \ likelihood \ with \ split \ reversed}{Data \ likelihood \ with \ split} \]
- No loss in accuracy when 50% of the splits are reversed.
Adaptive Splitting Results

<table>
<thead>
<tr>
<th>Model</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>88.4</td>
</tr>
<tr>
<td>With 50% Merging</td>
<td>89.5</td>
</tr>
</tbody>
</table>
Number of Phrasal Subcategories
## Final Results

<table>
<thead>
<tr>
<th>Parser</th>
<th>$F1 \leq 40$ words</th>
<th>$F1$ all words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Klein &amp; Manning ’03</td>
<td>86.3</td>
<td>85.7</td>
</tr>
<tr>
<td>Matsuzaki et al. ’05</td>
<td>86.7</td>
<td>86.1</td>
</tr>
<tr>
<td>Collins ’99</td>
<td>88.6</td>
<td>88.2</td>
</tr>
<tr>
<td>Charniak &amp; Johnson ’05</td>
<td>90.1</td>
<td>89.6</td>
</tr>
<tr>
<td>Petrov et. al. 06</td>
<td>90.2</td>
<td>89.7</td>
</tr>
</tbody>
</table>
Hierarchical Pruning
Parse multiple times, with grammars at different levels of granularity!

coarse:

split in two:

split in four:

split in eight:
1621 min
111 min
35 min
15 min [91.2 F1] (no search error)