Two views of linguistic structure:
1. Constituency (phrase structure)
   • Phrase structure organizes words into nested constituents.
   • Fed raises interest rates

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/pronoun:
  • I sat [on the box/right on top of the box/there].
Headed phrase structure

- Context-free grammar
- VP → ... VB* ...
- NP → ... NN* ...
- ADJP → ... JJ* ...
- ADVP → ... RB* ...
- S → ... NP VP ...
- Plus minor phrase types:
  - QP (quantifier phrase in NP), CONJP (multi-word constructions: as well as), INTJ (interjections), etc.

Two views of linguistic structure:
2. Dependency structure

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

The boy put the tortoise on the rug

Phrase Chunking

- Find all non-recursive noun phrases (NPs) and verb phrases (VPs) in a sentence.
  - [NP I] [VP ate] [NP the spaghetti] [PP with] [NP meatballs].
  - [NP He] [VP reckons] [NP the current account deficit] [VP will narrow] [PP to] [NP only 1.8 billion] [PP in] [NP September].

Phrase Chunking as Sequence Labeling

- Tag individual words with one of 3 tags
  - B (Begin) word starts new target phrase
  - I (Inside) word is part of target phrase but not the first word
  - O (Other) word is not part of target phrase
- Sample for NP chunking
  - He reckons the current account deficit will narrow to only 1.8 billion in September.
    Begin Inside Other

Evaluating Chunking

Per token accuracy does not evaluate finding correct full chunks. Instead use:

Precision = Number of correct chunks found / Total number of chunks found
Recall = Number of correct chunks found / Total number of actual chunks

F measure: \( F_\beta = \frac{1}{\left(\frac{1}{\beta^2} + 1\right)^{1/2}} \cdot \frac{2PR}{P + R} \)
Current Chunking Results

- Best system for NP chunking: $F_1 = 96\%$
- Typical results for finding range of chunk types (CONLL 2000 shared task: NP, VP, PP, ADV, SBAR, ADJP) is $F_1 = 92\% – 94\%$

Syntactic Parsing

- Produce the correct syntactic parse tree for a sentence.

Classical NLP Parsing: The problem and its solution

- Adding constraints to grammars to limit unlikely/weird parses for sentences
  - But the attempt make 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)

Statistical Parsing

- The correct syntactic parse tree for a sentence.

The rise of annotated data: The Penn Treebank

- 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

Two problems to solve for parsing:

1. Repeated work...
Two problems to solve for parsing:
1. Repeated work...

“Cats scratch people with cats with claws”

Two problems to solve for parsing:
2. Choosing the correct parse

• How do we work out the correct attachment:
  • She saw the man with a telescope
  • 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.

Statistical parsing applications

Statistical parsers are now robust and widely used in larger NLP applications:
• High precision question answering [Pasca and Heneghys SIGIR 2001]
• Improving biological named entity finding [Pielat et al. PAKIPS 2004]
• Syntactically based sentence compression [Lin and Wilbur 2007]
• Extracting opinions about products [Bloom et al. NAACL 2007]
• Improved interaction in computer games [Garrick and Roy 2003]
• Helping linguists find data [Hendek et al. BLS 2000]
• Source sentence analysis for machine translation [Xu et al. 2000]
• Relation extraction systems [Fundel et al. Bioinformatics 2006]

(Probabilistic) Context-Free Grammars

• CFG
• PCFG

Phrase structure grammars = context-free grammars (CFGs)

• $G = \{T, N, S, R\}$
  • $T$ 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 $X \rightarrow \gamma$
    • $X \in N$ and $\gamma \in (N \cup T)^*$

A phrase structure grammar

\[
\begin{align*}
S & \rightarrow NP \ VP \\
VP & \rightarrow V \ NP \\
NP & \rightarrow NP \ PP \\
NP & \rightarrow N \\
NP & \rightarrow e \\
PP & \rightarrow P \ NP \\
people & \rightarrow people \\
fish & \rightarrow fish \\
with & \rightarrow with \\
\end{align*}
\]

people fish tanks
people fish with rods
N -> people
N -> fish
N -> tanks
N -> rods
V -> people
V -> fish
V -> tanks
P -> with
Phrase structure grammars
= context-free grammars (CFGs)

• \( G = (T, N, S, R) \)
  • \( T \) 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 \( X \rightarrow \gamma \)
    • \( X \in N \) and \( \gamma \in (N \cup T)^* \)
• A grammar \( G \) generates a language \( L \).

Sentence Generation
• Sentences are generated by recursively rewriting the start symbol using the productions until only terminals symbols remain.

A phrase structure grammar

\[
\begin{align*}
S & \rightarrow NP \ VP \\
NP & \rightarrow V \ NP \\
NP & \rightarrow V \ NP \ PP \\
NP & \rightarrow N \\
NP & \rightarrow e \\
PP & \rightarrow P \ NP \\
VT & \rightarrow block \\
NP & \rightarrow the \\
NP & \rightarrow Nominal \\
NP & \rightarrow flight \\
PP & \rightarrow through \\
NP & \rightarrow Proper-NN \\
NP & \rightarrow Houston
\end{align*}
\]

Probabilistic – or stochastic – context-free grammars (PCFGs)

• \( G = (T, N, S, R, P) \)
  • \( T \) 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
    • \( P(X \rightarrow \gamma) \geq 0 \)
    • \( \sum_{\gamma \in R(X)} P(X \rightarrow \gamma) = 1 \)
• A grammar \( G \) generates a language model \( L \).

A PCFG

\[
\begin{align*}
S & \rightarrow NP \ VP & 1.0 & N \rightarrow people & 0.5 \\
VP & \rightarrow V \ NP & 0.6 & N \rightarrow fish & 0.2 \\
VP & \rightarrow V \ NP \ PP & 0.4 & N \rightarrow tanks & 0.2 \\
NP & \rightarrow NP \ NP & 0.1 & N \rightarrow rods & 0.1 \\
NP & \rightarrow NP \ PP & 0.2 & V \rightarrow people & 0.1 \\
NP & \rightarrow N & 0.7 & V \rightarrow fish & 0.6 \\
NP & \rightarrow e & 0.1 & V \rightarrow tanks & 0.3 \\
PP & \rightarrow P \ NP & 1.0 & P \rightarrow with & 1.0
\end{align*}
\]

[With empty NP removed to less ambiguous]
The probability of trees and strings

- $P(t)$ - The probability of a tree $t$ is the product of the probabilities of the rules used to generate it.
- $P(s)$ - The probability of the string $s$ is the sum of the probabilities of the trees which have that string as their yield
  \[ P(s) = \sum_t P(s, t) \text{ where } t \text{ is a parse of } s \]

Tree and String Probabilities

- $s = \text{people fish tanks with rods}$
- $P(t_1) = 0.0008232$
- $P(t_2) = 0.00024696$
- $P(s) = P(t_1) + P(t_2) = 0.00107016$

Chomsky Normal Form

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

A phrase structure grammar

<table>
<thead>
<tr>
<th>Production</th>
<th>Nonterminal</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow NP \ VP$</td>
<td>$N \rightarrow \text{people}$</td>
</tr>
<tr>
<td>$VP \rightarrow V \ NP$</td>
<td>$N \rightarrow \text{fish}$</td>
</tr>
<tr>
<td>$VP \rightarrow V$</td>
<td>$N \rightarrow \text{tanks}$</td>
</tr>
<tr>
<td>$NP \rightarrow NP \ PP$</td>
<td>$V \rightarrow \text{people}$</td>
</tr>
<tr>
<td>$NP \rightarrow N$</td>
<td>$V \rightarrow \text{fish}$</td>
</tr>
<tr>
<td>$NP \rightarrow P$</td>
<td>$V \rightarrow \text{tanks}$</td>
</tr>
<tr>
<td>$PP \rightarrow P \ NP$</td>
<td>$P \rightarrow \text{with}$</td>
</tr>
</tbody>
</table>

Chomsky Normal Form steps

<table>
<thead>
<tr>
<th>Production</th>
<th>Nonterminal</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow NP \ VP$</td>
<td>$N \rightarrow \text{people}$</td>
</tr>
<tr>
<td>$S \rightarrow VP$</td>
<td>$N \rightarrow \text{fish}$</td>
</tr>
<tr>
<td>$VP \rightarrow V \ NP$</td>
<td>$N \rightarrow \text{tanks}$</td>
</tr>
<tr>
<td>$VP \rightarrow V$</td>
<td>$N \rightarrow \text{rods}$</td>
</tr>
<tr>
<td>$NP \rightarrow NP \ PP$</td>
<td>$V \rightarrow \text{people}$</td>
</tr>
<tr>
<td>$NP \rightarrow NP$</td>
<td>$V \rightarrow \text{fish}$</td>
</tr>
<tr>
<td>$NP \rightarrow P$</td>
<td>$V \rightarrow \text{tanks}$</td>
</tr>
<tr>
<td>$PP \rightarrow P \ NP$</td>
<td>$P \rightarrow \text{with}$</td>
</tr>
<tr>
<td>$PP \rightarrow P$</td>
<td>$P \rightarrow \text{with}$</td>
</tr>
</tbody>
</table>
Chomsky Normal Form steps

\[
S \rightarrow NP \ V P \\
NP \rightarrow N \ P N P \\
NP \rightarrow N \ V N \\
V P \rightarrow V \ NP P \\
V P \rightarrow V \ NP \\
S \rightarrow NP \ V P \\
S \rightarrow V \ NP \\
S \rightarrow V \ NP N \\
P N P \rightarrow N \ P N P \\
N P P \rightarrow N \ P P \\
V \ P P \rightarrow V \ P P \\
P \ P P \rightarrow P \ P P \\
N \rightarrow people \\
N \rightarrow fish \\
N \rightarrow tanks \\
N \rightarrow rods \\
V \rightarrow people \\
V \rightarrow fish \\
V \rightarrow tanks \\
P \rightarrow with
\]

Chomsky Normal Form steps

\[
S \rightarrow NP \ V P \\
NP \rightarrow N \ P N P \\
NP \rightarrow N \ V N \\
V P \rightarrow V \ NP P \\
V P \rightarrow V \ NP \\
S \rightarrow NP \ V P \\
S \rightarrow V \ NP \\
S \rightarrow V \ NP N \\
P N P \rightarrow N \ P N P \\
N P P \rightarrow N \ P P \\
V \ P P \rightarrow V \ P P \\
P \ P P \rightarrow P \ P P \\
N \rightarrow people \\
N \rightarrow fish \\
N \rightarrow tanks \\
N \rightarrow rods \\
V \rightarrow people \\
V \rightarrow fish \\
V \rightarrow tanks \\
P \rightarrow with
\]

You should think of this as a transformation for efficient parsing

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...

Before and After binarization on VP

Parsing

• Given a string of terminals (e.g. sentences) and a CFG, determine if the string can be generated by the CFG.
• Also return a parse tree for the string
• Also return all possible parse trees for the string
• Must search space of derivations for one that derives the given string.
  • Top-Down Parsing: Start searching space of derivations for the start symbol.
  • Bottom-up Parsing: Start search space of reverse derivations from the terminal symbols in the string.

Parsing Example

Top Down Parsing
Top Down Parsing

S
NP VP
Proper/Noun

Top Down Parsing

S
NP VP
Proper/Noun
X
book

Top Down Parsing

S
NP VP
Det Nominal

Top Down Parsing

S
NP VP
Det Nominal
X
book

Top Down Parsing

S
Aux NP VP

Top Down Parsing

S
Aux NP VP
X
book
Top Down Parsing

S
| VP

Top Down Parsing

S
| VP
| Verb
| book

Top Down Parsing

S
| VP
| Verb
| book
| that

Top Down Parsing

S
| VP
| Verb
| NP
| book

Top Down Parsing

S
| VP
| Verb
| NP
| book
Bottom Up Parsing

```
book that flight
```

Bottom Up Parsing

```
Nominal
Nominal
Nominal
X
Noun
book that flight
```

Bottom Up Parsing

```
Nominal
Nominal
Noun
PP
book that flight
```

Bottom Up Parsing

```
Nominal
Nominal
Noun
Det NP
book that flight
```

Bottom Up Parsing

```
Nominal
Nominal
Noun
Det NP
book that flight
```

Bottom Up Parsing

```
Nominal
Nominal
Noun
Det NP
book that flight
```

Bottom Up Parsing

```
Nominal
Nominal
Noun
Det NP
book that flight
```
Bottom Up Parsing

Nominal
  PP
    S
      VP
        NP
          Det
            that
              Nominal
                flight
              Nominal
                book
          Nominal
                flight
              Nominal
                book

Bottom Up Parsing

Nominal
  NP
    Det
      that
        Nominal
          Nominal
            flight
          Nominal
            book

Bottom Up Parsing

Nominal
  PP
    S
      VP
        NP
          Det
            that
              Nominal
                flight
              Nominal
                book
          Nominal
                flight
              Nominal
                book

Bottom Up Parsing

Nominal
  VP
    S
      NP
        Det
          that
            Nominal
              Nominal
                flight
              Nominal
                book

Bottom Up Parsing

Nominal
  VP
    S
      NP
        Det
          that
            Nominal
              Nominal
                flight
              Nominal
                book
Bottom Up Parsing

```
S
  VP
    book
  X
    NP
    Det
    that
    Nominal
    flight
```

Bottom Up Parsing

```
NP
  Det
  that
  Nominal
  flight
```

Bottom Up Parsing

```
VP
  book
```

Bottom Up Parsing

```
S
  VP
    book
  Det
    that
    Nominal
    flight
```

Bottom Up Parsing

```
NP
  Nominal
```

Bottom Up Parsing

```
VP
  Nominal
  flight
```

Bottom Up Parsing

```
S
  VP
    book
  Det
    that
    Nominal
    flight
```
Top Down vs. Bottom Up

- Top down never explores options that will not lead to a full parse, but can explore many options that never connect to the actual sentence.
- Bottom up never explores options that do not connect to the actual sentence but can explore options that can never lead to a full parse.
- Relative amounts of wasted search depend on how much the grammar branches in each direction.

Two problems to solve for parsing:
1. Repeated work
   - "Cats scratch people with cats with claws"

Dynamic Programming Parsing

- To avoid extensive repeated work, must cache intermediate results, i.e. completed phrases.
- Caching (memorizing) is critical to obtaining a polynomial time parsing (recognition) algorithm for CFGs.

(Probabilistic) CKY Parsing

Constituency Parsing

Input: a PCFG, and a sentence

Output: a parsing tree
Extended CKY parsing

- Unaries can be incorporated into the algorithm
- Messy, but doesn’t increase algorithmic complexity
- Empties can be incorporated
- 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

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-B-C)
        if prob > score[begin][end][A]
          score[begin][end][A] = prob
          back[begin][end][A] = new Triple(split,B,C)
          added = true
      //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)
```