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.
    - I sat [on the box/right on top of the box/here].

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: some people), CONJP (multi-word constructions: as well as), INTJ (interjections: aha), 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 [VP [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

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:

\[ F = \frac{2 \times P \times R}{P + R} \]

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, ADJ, SBAR, ADJP) is \( F_1 = 92\% - 94\% \)
Syntactic Parsing

• Produce 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...

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 Herbst SIGIR 2001]
- Improving biological named entity finding [Pietal et al. JPLIA 2004]
- Syntactically based sentence compression [Lin and Wilbur 2007]
- Extracting opinions about products [Bloom et al. NAACL 2007]
- Improved interaction in computer games [Carnak and Roy 2005]
- Helping linguists find data [Resnik et al. BLS 2005]
- 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 ∈ N)
  - R is a set of rules/productions of the form X → γ
    * X ∈ N and γ ∈ (N ∪ T)*

A phrase structure grammar

\[
S \rightarrow NP \ VP \\
VP \rightarrow V \ NP \\
VP \rightarrow V \ NP \ PP \\
NP \rightarrow NP \ NP \\
NP \rightarrow N \\
NP \rightarrow e \\
PP \rightarrow P \ NP \\
\]

people fish tanks
people fish with rods

Sentence Generation

- Sentences are generated by recursively rewriting the start symbol using the productions until only terminals symbols remain.
Phrase structure grammars in NLP

- \( G = (T, C, N, S, L, R) \)
- \( T \) 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 \( X \xrightarrow{\phi} x \)
  - \( X \in C \) and \( x \in T \)
- \( R \) is the grammar, a set of items of the form \( X \xrightarrow{g} \)
  - \( X \in N \) and \( g \in (N \cup C)^* \)
- 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 phrase structure grammar

- \( S \rightarrow NP \ VP \)
- \( VP \rightarrow V \ NP \)
- \( NP \rightarrow NP \ PP \)
- \( NP \rightarrow N \)
- \( NP \rightarrow e \)
- \( PP \rightarrow P \ NP \)

people fish tanks
people fish with rods

A PCFG

- \( S \rightarrow NP \ VP \ 1.0 \)
- \( VP \rightarrow V \ NP \ 0.6 \)
- \( VP \rightarrow V \ NP \ PP \ 0.4 \)
- \( NP \rightarrow NP \ NP \ 0.1 \)
- \( NP \rightarrow NP \ PP \ 0.2 \)
- \( NP \rightarrow N \ 0.7 \)
- \( PP \rightarrow P \ NP \ 1.0 \)

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 \( X \xrightarrow{g} \)
- \( P \) is a probability function
  - \( P \rightarrow [0,1] \)
  - \( \forall X \in N, \sum_{y} P(X \rightarrow y) = 1 \)
- A grammar \( G \) generates a language model \( L \).

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

\[
P(s) = \sum_{t} P(s,t) \] where \( t \) is a parse of \( s \)

\[
P(s) = \sum_{t} P(t) \]

[With empty NP removed to fine-tune parse]
Tree and String Probabilities

- \( s = \) people fish tanks with rods
- \( P(t_1) = 1.0 \times 0.7 \times 0.4 \times 0.5 \times 0.6 \times 0.7 \times 1.0 \times 0.2 \times 1.0 \times 0.7 \times 0.1 = 0.0008232 \)
- \( P(t_2) = 1.0 \times 0.7 \times 0.6 \times 0.5 \times 0.6 \times 0.2 \times 0.7 \times 1.0 \times 0.2 \times 1.0 \times 0.7 \times 0.1 = 0.00024696 \)
- \( P(s) = P(t_1) + P(t_2) = 0.0008232 + 0.00024696 = 0.00107016 \)

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 unaries are removed recursively
- \( n \)-ary rules are divided by introducing new nonterminals (\( n > 2 \))

A phrase structure grammar

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

\[
\begin{align*}
N & \rightarrow \text{people} \\
N & \rightarrow \text{fish} \\
N & \rightarrow \text{tanks} \\
N & \rightarrow \text{rods} \\
V & \rightarrow \text{people} \\
V & \rightarrow \text{fish} \\
V & \rightarrow \text{tanks} \\
P & \rightarrow \with
\end{align*}
\]
Chomsky Normal Form steps

S → NP VP
VP → V NP
S → VP
VP → V NP PP
S → V VP PP
VP → V PP
S → V PP
VP → V
S → NP
NP → NP
NP → PP
NP → N
NP → PP
PP → P
PP → N
V → people
V → fish
V → tanks
V → rods
P → with

Chomsky Normal Form steps

S → NP VP
VP → V NP
S → VP
VP → V NP PP
S → V NP PP
VP → V PP
S → V PP
VP → V
S → NP
NP → NP
NP → PP
NP → P NP
PP → P NP
PP → N
V → people
V → fish
V → tanks
V → rods
P → with

An example: before binarization...

Before and After binarization on VP

• 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
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

```
S
  | NP  VP
  | Pronoun

S
  | NP  VP
  | ProperNoun

S
  | NP  VP
  | Pronoun
  | book
```

```
S
  | NP  VP
  | book
```

```
S
  | NP  VP
  | that
```

```
S
  | NP  VP
  | that
```

```
S
  | NP  VP
  | flight
```

```
S
  | NP  VP
  | flight
```

```
S
  | NP  VP
  | book that flight
```

```
S
  | NP  VP
  | book that flight
```

```
S
  | NP  VP
  | book that flight
```

```
S
  | NP  VP
  | book that flight
```

```
S
  | NP  VP
  | book that flight
```

```
S
  | NP  VP
  | book that flight
```

```
S
  | NP  VP
  | book that flight
```

```
S
  | NP  VP
  | book that flight
```

```
S
  | NP  VP
  | book that flight
```

```
S
  | NP  VP
  | book that flight
```

```
S
  | NP  VP
  | book that flight
```
Top Down Parsing

```
S  
|   
NP  VP
  
Det  Nominal
```

Top Down Parsing

```
S  
|   
NP  VP
  
Det  Nominal

book
```

Top Down Parsing

```
S  
|   
NP  VP
  
Aux  NP  VP

Verb
```

Top Down Parsing

```
S  
|   
NP  VP
  
Det  Nominal

book
```

Top Down Parsing

```
S  
|   
NP  VP
  
Aux  NP  VP

Verb
```
Top Down Parsing

S

| VP |

| Verb |

| book |

Top Down Parsing

S

| VP |

| Verb |

| X |

| book |

Top Down Parsing

S

| VP |

| Verb |

| NP |

| book |

Top Down Parsing

S

| VP |

| Verb |

| NP |

| Pronoun |

| book |

Top Down Parsing

S

| VP |

| Verb |

| NP |

| Pronoun |

| X |

| that |
Top Down Parsing

[Diagram]

Top Down Parsing

[Diagram]

Top Down Parsing

[Diagram]

Top Down Parsing

[Diagram]

Top Down Parsing

[Diagram]

Top Down Parsing

[Diagram]
Bottom Up Parsing

\[
\begin{array}{c}
\text{VP} \\
\text{VP} \\
\text{Verb} \\
\text{book} \\
\text{Det} \\
\text{that} \\
\text{Nominal} \\
\text{flight} \\
\end{array}
\]

Bottom Up Parsing

\[
\begin{array}{c}
\text{VP} \\
\text{NP} \\
\text{Verb} \\
\text{book} \\
\text{Det} \\
\text{that} \\
\text{Nominal} \\
\text{flight} \\
\end{array}
\]

Bottom Up Parsing

\[
\begin{array}{c}
\text{VP} \\
\text{S} \\
\text{Verb} \\
\text{Det} \\
\text{that} \\
\text{Nominal} \\
\text{flight} \\
\end{array}
\]

Bottom Up Parsing

\[
\begin{array}{c}
\text{VP} \\
\text{NP} \\
\text{Verb} \\
\text{book} \\
\text{Det} \\
\text{that} \\
\text{Nominal} \\
\text{flight} \\
\end{array}
\]

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

PCFG

Rule Prob θ
S → NP VP 0.9
NP → NP VP 0.5
NP → V NP 0.3
VP → V 0.1

Viterbi (Max) Scores

S → NP VP 0.9
S → VP 0.1
VP → V NP 0.3
VP → V 0.1
V = V @VP_V 0.3
@VP_V → NP PP 1.0
NP → NP PP 0.2
NP → NP 0.2
PP → P NP 1.0
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

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 not in grammar
        score[i][i+1][A] = P(A)
    //handle unaries
  while added
    added = false
    for A, B in nonterms
      if score[i][i+1][B] > 0 && A -> B in grammar
        prob = P(A) * 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
      added = false
      for A, B in nonterms
        if score[i][i+1][B] > 0 && A -> B in grammar
          prob = P(A) * 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
  return buildTree(score, back)