CS 6120/CS4120: Natural Language Processing

Instructor: Prof. Lu Wang
College of Computer and Information Science
Northeastern University
Webpage: www.ccs.neu.edu/home/luwang
Two views of linguistic structure:
1. Constituency (phrase structure)

• Phrase structure organizes words into nested constituents.
  • Fed raises interest rates
Two views of linguistic structure:
1. Constituency (phrase structure)
   - Phrase structure organizes words into nested constituents.
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/pronoun:
    • I sat [on the box/right on top of the box/there].
Analysts said -NONE- 0 NP-SBJ-1 VP
  NNP NNP VBV S
  Mr. Stronach wants NP-SBJ VP
  -NONE- TO VP
  ~-1 to VB NP
  resume

DT ADJP NN IN S-NOM
  a RBR JJ role in NP-SBJ VP
  more influential

-VBG NP
  * running DT NN
  the company
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*
Two views of linguistic structure:

2. Dependency structure

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

<table>
<thead>
<tr>
<th>Begin</th>
<th>Inside</th>
<th>Other</th>
</tr>
</thead>
</table>

Evaluating Chunking

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

Precision = \frac{\text{Number of correct chunks found}}{\text{Total number of chunks found}}

Recall = \frac{\text{Number of correct chunks found}}{\text{Total number of actual chunks}}

F measure: 
\begin{align*}
F_1 \ &= \ \frac{1}{\frac{1}{P} + \frac{1}{R}}/2 \\
&= \frac{2PR}{P + R}
\end{align*}
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.
The rise of annotated data: The Penn Treebank

[Marcus et al. 1993, Computational Linguistics]
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
Two problems to solve for parsing:
1. Repeated work...

“Cats scratch people with cats with claws”
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 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]
(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

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

people fish tanks
people fish with rods
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.
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 $X \rightarrow x$
    • $X \in C$ and $x \in T$
  • $R$ 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 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

people fish tanks
people fish with rods
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
    • $P: R \rightarrow [0,1]$
    • $\forall X \in N, \sum_{X \rightarrow \gamma \in R} P(X \rightarrow \gamma) = 1$

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

S → NP VP  1.0
VP → V NP  0.6
VP → V NP PP  0.4
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

[With empty NP removed so 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.

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

\[ = \sum_t P(t) \]
$t_1$: 

- S$_{1.0}$
  - NP$_{0.7}$
    - N$_{0.5}$: people
    - V$_{0.6}$: fish
  - VP$_{0.4}$
    - NP$_{0.7}$
      - N$_{0.2}$: tanks
      - P$_{1.0}$: with
      - NP$_{0.7}$
        - N$_{0.1}$: rods

$t_2$: 

- S$_{1.0}$
  - NP$_{0.7}$
    - N$_{0.5}$: people
    - V$_{0.6}$: fish
  - VP$_{0.6}$
    - NP$_{0.2}$
      - N$_{0.2}$: tanks
      - P$_{1.0}$: with
      - NP$_{0.7}$
        - N$_{0.1}$: rods
Tree and String Probabilities

• $s = \textit{people fish tanks with rods}$

• $P(t_1) = 1.0 \times 0.7 \times 0.4 \times 0.5 \times 0.6 \times 0.7 \times 1.0 \times 0.2 \times 1.0 \times 0.7 \times 0.1$
  
  $= 0.0008232$

• $P(t_2) = 1.0 \times 0.7 \times 0.6 \times 0.5 \times 0.6 \times 0.2 \times 0.7 \times 1.0 \times 0.2 \times 1.0 \times 0.7 \times 0.1$
  
  $= 0.00024696$

• $P(s) = P(t_1) + P(t_2)$
  
  $= 0.0008232 + 0.00024696$

  $= 0.00107016$
Chomsky Normal Form

• All rules are of the form X → Y Z or X → w
  • X, Y, Z ∈ N and w ∈ 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

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

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

\[ N \rightarrow people \]
\[ N \rightarrow fish \]
\[ N \rightarrow tanks \]
\[ N \rightarrow rods \]
\[ V \rightarrow people \]
\[ V \rightarrow fish \]
\[ V \rightarrow tanks \]
\[ P \rightarrow with \]
Chomsky Normal Form steps

S \rightarrow NP \ VP
VP \rightarrow V \ NP
S \rightarrow V \ NP
VP \rightarrow V
VP \rightarrow V \ NP \ PP
S \rightarrow V \ NP \ PP
VP \rightarrow V \ PP
S \rightarrow V \ PP
NP \rightarrow NP \ NP
NP \rightarrow NP
NP \rightarrow NP \ PP
NP \rightarrow PP
NP \rightarrow N
PP \rightarrow P \ NP
PP \rightarrow P

N \rightarrow people
N \rightarrow fish
N \rightarrow tanks
N \rightarrow rods
V \rightarrow people
S \rightarrow people
V \rightarrow fish
S \rightarrow fish
V \rightarrow tanks
S \rightarrow tanks
P \rightarrow 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

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

\[
\begin{align*}
NP &\rightarrow people \\
NP &\rightarrow fish \\
NP &\rightarrow tanks \\
NP &\rightarrow rods \\
V &\rightarrow people \\
S &\rightarrow people \\
VP &\rightarrow people \\
V &\rightarrow fish \\
S &\rightarrow fish \\
VP &\rightarrow fish \\
V &\rightarrow tanks \\
S &\rightarrow tanks \\
VP &\rightarrow tanks \\
P &\rightarrow with \\
PP &\rightarrow with
\end{align*}
\]
Chomsky Normal Form

• 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

Before binarization:
- **NP**
  - people
  - fish
  - tanks
  - with
  - rods

After binarization:
- **NP**
  - people
  - fish
  - tanks
  - with
  - rods
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

book that flight

S
  /   
VP  
    /   
Verb NP
      /  
book Det Nominal
         /  
that Noun
             / 
               flight
Top Down Parsing

S

NP   VP

Pronoun
Top Down Parsing

```
S
 /   |
NP   VP
   /   |
Pronoun
   /   |
X   book
```
Top Down Parsing

S
  /\  \
/  \ /  \  
NP  VP
  
ProperNoun
Top Down Parsing

S
  NP  VP
  /
ProperNoun
  /
    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
Top Down Parsing

S
| VP
Top Down Parsing

S
/|
VP
/|
Verb
Top Down Parsing

S
  /\
 VP
  /\
Verb
  /\
book
Top Down Parsing
Top Down Parsing

```
S
  /
 VP
  /
 Verb NP
```
Top Down Parsing

S
  /\
VP
  /\ Verb NP
   /\
  book
Top Down Parsing

```
S
  /\      
VP  
  /\     
Verb NP
  /\     
book Pronoun
```
Top Down Parsing

```
S
  /
VP
   /
Verb NP
   book Pronoun
      X
       that
```
Top Down Parsing

S
  /
VP
  /  
Verb NP
    /  
book ProperNoun
Top Down Parsing

```
S
  |
  VP
    |
    Verb NP
       |
       book ProperNoun
          |
          X
              |
              that
```
Top Down Parsing

```
S
  /
VP
  /
  Verb NP
  /
  book Det Nominal
```
Top Down Parsing

```
S
 /  
VP
   /  
Verb NP
   /  
book Det Nominal
   /  
that
```
Top Down Parsing

```
S
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>VP</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Verb NP</td>
</tr>
<tr>
<td>book Det Nominal</td>
</tr>
<tr>
<td>that Noun</td>
</tr>
</tbody>
</table>
```
Top Down Parsing

```
S
  /\  
 VP
  / \  
 Verb NP
     / \  
    book Det Nominal
         / \
        that Noun
                 \
                  flight
```
Bottom Up Parsing

book    that    flight
Bottom Up Parsing

Noun
book that flight
Bottom Up Parsing

Nominal
   |
   Noun
   |
book   that   flight
Bottom Up Parsing

Nominal

Nominal  Noun

Noun

book  that  flight
Bottom Up Parsing

```
Nominal
    /\     /
  Nominal  Noun
    |     /\  
  Noun  book  that  flight
```
Bottom Up Parsing

```
Nominal
  Nominal  PP
    Noun
    book  that  flight
```
Bottom Up Parsing
Bottom Up Parsing

```
Nominal
  /     \
Nominal PP
    /   |
Noun  Det  Nominal
  book  that  flight
```
Bottom Up Parsing

```
Nominal
  /   
Nominal PP
    / 
  Noun  NP
     /  
    that Nominal
       /     
      Noun  flight
```
Bottom Up Parsing
Bottom Up Parsing
Bottom Up Parsing
Bottom Up Parsing

Nominal
  Nominal
    Noun
      book
  PP
    Det
      that
  Nominal
    Noun
      flight
Bottom Up Parsing

```
NP
  /\   \
Verb book
  /\   /\   \\
Det that Nominal \\
  /\   \\
Noun flight
```
Bottom Up Parsing

\[
\begin{align*}
\text{VP} & \quad \text{NP} \\
\text{Verb} & \quad \text{Det} \quad \text{Nominal} \\
\text{book} & \quad \text{that} \quad \text{Noun} \\
& \quad \text{flight}
\end{align*}
\]
Bottom Up Parsing

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

```
VP
├── VP
│   ├── Verb
│       └── book
├── PP
│   └── Det
│       └── that
└── NP
    └── Nominal
        └── Noun
            └── flight
```
Bottom Up Parsing
Bottom Up Parsing
Bottom Up Parsing

```
VP
  Verb: book
  Det: that
  Nominal: flight
```

```
NP
  Noun: flight
```
Bottom Up Parsing

```
S
  /\  
VP  NP
  /  
Verb Det Nominal
  /    
book that Noun
      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

fish  people  fish  tanks

PCFG

Rule Prob $\theta_i$

$S \rightarrow \text{NP VP} \quad \theta_0$

$\text{NP} \rightarrow \text{NP NP} \quad \theta_1$

$\ldots$

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

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

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

$\ldots$
Constituency Parsing

Output: a parsing tree

PCFG

<table>
<thead>
<tr>
<th>Rule Prob $\theta_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow NP \ VP$</td>
</tr>
<tr>
<td>$NP \rightarrow NP \ NP$</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>$N \rightarrow fish$</td>
</tr>
<tr>
<td>$N \rightarrow people$</td>
</tr>
<tr>
<td>$V \rightarrow fish$</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Cocke-Kasami-Younger (CKY) Constituency Parsing
Viterbi (Max) Scores

NP → NP VP 0.35
NP → NP PP 0.14
NP → N 0.5

VP → V NP 0.1
VP → V @VP_V 0.1
VP → V PP 0.1
@VP_V → NP PP 1.0

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
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 \rightarrow words[i] in grammar
            score[i][i+1][A] = P(A \rightarrow words[i])
//handle unaries
boolean added = true
while added
    added = false
    for A, B in nonterms
        if score[i][i+1][B] > 0 && A \rightarrow B in grammar
            prob = P(A \rightarrow 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
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)