CS 6120/CS4120: Natural Language Processing

Instructor: Prof. Lu Wang
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 Mr. Stronach wants to resume a more influential role in 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.

  Begin    Inside    Other
Evaluating Chunking

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

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

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

F measure: \[ F_1 = \frac{1}{\left(\frac{1}{P} + \frac{1}{R}\right)/2} = \frac{2PR}{P + R} \]
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.
Annotated data: The Penn Treebank [Marcus et al. 1993, Computational Linguistics]

(S
  (NP-SBJ (DT The) (NN move))
  (VP (VBD followed)
    (NP
      (NP (DT a) (NN round))
      (PP (IN of)
        (NP
          (NP (JJ similar) (NNS increases))
          (PP (IN by)
            (NP (JJ other) (NNS lenders)))
          (PP (IN against)
            (NP (NNP Arizona) (JJ real) (NN estate) (NNS loans)))))))
  (, ,))
(S-ADV
  (NP-SBJ (-NONE- *))
  (VP (VBG reflecting)
    (NP
      (NP (DT a) (VBG continuing) (NN decline))
      (PP-LOC (IN in)
        (NP (DT that) (NN market)))))))
(. .)))

[73x468]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

<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 → V NP</td>
<td>0.6</td>
</tr>
<tr>
<td>VP → V NP PP</td>
<td>0.4</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>

[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

\[
P(s) = \sum_t P(s, t) \quad \text{where } t \text{ is a parse of } s
\]
\[
= \sum_t P(t)
\]
\( t_1: \)

\[
S_{1.0} \\
\quad NP_{0.7} \\
\quad \quad N_{0.5} \quad V_{0.6} \quad NP_{0.7} \quad PP_{1.0} \\
\quad \quad \quad people \quad fish \quad tanks \quad with \quad N_{0.1} \\
\quad \quad \quad \quad rods
\]

\( t_2: \)

\[
S_{1.0} \\
\quad NP_{0.7} \\
\quad \quad N_{0.5} \quad V_{0.6} \quad NP_{0.7} \quad PP_{1.0} \\
\quad \quad people \quad fish \quad tanks \quad with \quad N_{0.1} \\
\quad \quad rods
\]
Tree and String Probabilities

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

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

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

• \( P(s) = P(t_1) + P(t_2) \)

  \[
  P(s) = 0.0008232 + 0.00024696 = 0.00107016
  \]
Chomsky Normal Form

• All rules are of the form $X \rightarrow YZ$ 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

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

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 → 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
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
S → people
V → fish
S → fish
V → tanks
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

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

• **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:

ROOT
S
NP
V
NP
PP
N
V
PP
N
N
people
fish
tanks
with
rods

After binarization:

ROOT
S
NP
VP
@VP_V
NP
V
NP
PP
N
N
N
N
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

```
S
  /\  \
Aux NP VP
  /  \\
 X  book
```
Top Down Parsing

S
/  
VP
Top Down Parsing

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

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

S
  /  
VP
  /   
Verb
  /    
book  that
Top Down Parsing

S

/    

VP

Verb  NP
Top Down Parsing

```
  S
  |  
  V P
  |   
Verb NP
   |   
book
```
Top Down Parsing

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

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

Nominal
  └── Noun
    └── flight
Bottom Up Parsing

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

Nominal
  Nominal  PP
    Noun   Det
      book  that  flight
Bottom Up Parsing

Diagram:

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

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

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

[Diagram of Bottom Up Parsing with the structure: Verb (book) -> Det (that) -> Nominal (flight)]
Bottom Up Parsing
Bottom Up Parsing

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

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

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

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

```
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

PCFG

Rule Prob $\theta_i$

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

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

$\ldots$

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

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

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

$\ldots$

fish people fish tanks
Constituency Parsing

Output: a parsing tree

PCFG

Rule Prob $\theta_i$

$S \rightarrow NP \ VP \quad \theta_0$
$NP \rightarrow NP \ NP \quad \theta_1$
$N \rightarrow \text{fish} \quad \theta_{42}$
$N \rightarrow \text{people} \quad \theta_{43}$
$V \rightarrow \text{fish} \quad \theta_{44}$
Cocke-Kasami-Younger (CKY) Constituency Parsing

fish people fish tanks

S

NP

N N V N

fish people fish tanks
Reusing local decisions

NP → people 0.35
V → people 0.1
N → people 0.5
VP → fish 0.06
V → fish 0.6
N → fish 0.2
S → NP VP 0.9
S → VP 0.1
VP → V NP 0.5
NP → NP NP 0.1
NP → NP PP 0.2
PP → P NP 1.0
Reusing local decisions

NP → people 0.35
V → people 0.1
N → people 0.5
VP → fish 0.06
V → fish 0.6
N → fish 0.2

S → NP VP 0.9
S → VP 0.1
VP → V NP 0.5
NP → NP NP 0.1
NP → NP PP 0.2
PP → P NP 1.0
Reusing local decisions

S→NP VP 0.9*0.35*0.06

NP 0.35
V 0.1
N 0.5

VP 0.06
V 0.6
N 0.2

NP→people 0.35
V→people 0.1
N→people 0.5
VP→fish 0.06
V→fish 0.6
N→fish 0.2

S→NP VP 0.9
S→VP 0.1
VP→V NP 0.5
NP→NP NP 0.1
NP→NP PP 0.2
PP→P NP 1.0
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

```python
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)
```