Parts of speech can be thought of as the lowest level of syntactic information. Groups words together into categories.

likes to eat candy.

What can/can’t go here?

Admin

- Assignment 2
- Assignment 3
  - Technically due Sunday Oct. 16 at midnight
  - Work in pairs
  - Any programming language
  - Given example output

Constituency

- Nouns
determiner nouns
  - The man
  - The boy
  - The cat
- Pronouns
determiner nouns +
  - The man that I saw
  - The boy with the blue pants
  - The cat in the hat
Constituency

- Words in languages tend to form into functional groups (parts of speech)
- Groups of words (aka phrases) can also be grouped into functional groups
  - often some relation to parts of speech
  - though, more complex interactions
- These phrase groups are called constituents

Common constituents

The man in the hat ran to the park.

- noun phrase
- prepositional phrase
- prepositional phrase
- noun phrase
- verb phrase
A number of related problems:

- Given a sentence, can we determine the syntactic structure?
- Can we determine if a sentence is grammatical?
- Can we determine how likely a sentence is to be grammatical to be an English sentence?
- Can we generate candidate, grammatical sentences?
Grammars

What is a grammar (3rd grade again…)?

Grammar is a set of structural rules that govern the composition of sentences, phrases and words.

Lots of different kinds of grammars:
- regular
- context-free
- context-sensitive
- recursively enumerable
- transformation grammars

States

What is the capital of this state? Jefferson City (Missouri)

Context free grammar

- How many people have heard of them?
- Look like:

```
S → NP VP
```

left hand side right hand side
(single symbol) (one or more symbols)
Formally...

G = (NT, T, P, S)
- NT: finite set of nonterminal symbols
- T: finite set of terminal symbols, NT and T are disjoint
- P: finite set of productions of the form A → α, A ∈ V and α ∈ (T ∪ NT)^*
- S ∈ NT: start symbol

CFG: Example

Many possible CFGs for English, here is an example (fragment):
- S → NP VP
- VP → V NP
- NP → DetP N | AdjP NP
- AdjP → Adj | Adv AdjP
- N → boy | girl
- V → sees | likes
- Adj → big | small
- Adv → very
- DetP → a | the

Grammar questions

- Can we determine if a sentence is grammatical?
- Given a sentence, can we determine the syntactic structure?
- Can we determine how likely a sentence is to be grammatical? to be an English sentence?
- Can we generate candidate, grammatical sentences?

Which of these can we answer with a CFG? How?
Derivations in a CFG

\[
S \rightarrow \text{NP VP}
\]

\[
\text{NP} \rightarrow \text{DetP N} \mid \text{AdjP NP}
\]

\[
\text{AdjP} \rightarrow \text{Adj} \mid \text{Adv AdjP}
\]

\[
\text{N} \rightarrow \text{boy} \mid \text{girl}
\]

\[
\text{V} \rightarrow \text{sees} \mid \text{likes}
\]

\[
\text{Adv} \rightarrow \text{very}
\]

\[
\text{DetP} \rightarrow \text{a} \mid \text{the}
\]

Derivations in a CFG

\[
\text{S} \rightarrow \text{NP VP}
\]

\[
\text{NP} \rightarrow \text{DetP N} \mid \text{AdjP NP}
\]

\[
\text{AdjP} \rightarrow \text{Adj} \mid \text{Adv AdjP}
\]

\[
\text{N} \rightarrow \text{boy} \mid \text{girl}
\]

\[
\text{V} \rightarrow \text{sees} \mid \text{likes}
\]

\[
\text{Adj} \rightarrow \text{big} \mid \text{small}
\]

\[
\text{Adv} \rightarrow \text{very}
\]

\[
\text{DetP} \rightarrow \text{a} \mid \text{the}
\]
Derivations in a CFG

\[
S \to NP \ VP \\
NP \to DetP \ N \mid AdjP \ NP \\
VP \to V \ NP \\
AdjP \to Adj \mid Adv \ AdjP \\
N \to boy \mid girl \\
V \to sees \mid likes \\
Adj \to big \mid small \\
Adv \to very \\
DetP \to a \mid the \\
\]

the boy likes NP

Derivations in a CFG

\[
S \to NP \ VP \\
NP \to DetP \ N \mid AdjP \ NP \\
VP \to V \ NP \\
AdjP \to Adj \mid Adv \ AdjP \\
N \to boy \mid girl \\
V \to sees \mid likes \\
Adj \to big \mid small \\
Adv \to very \\
DetP \to a \mid the \\
\]

the boy likes a girl

Derivations of CFGs

- String rewriting system: we derive a string
- But derivation history represented by phrase-structure tree

the boy likes a girl

the boy likes a girl

DetP \to a \mid the
 Parsing

- Parsing is the field of NLP interested in automatically determining the syntactic structure of a sentence.
- Parsing can be thought of as determining what sentences are “valid” English sentences.
- As a by product, we often can get the structure.

Given a CFG and a sentence, determine the possible parse tree(s)

I eat sushi with tuna

What parse trees are possible for this sentence?

What if the grammar is much larger?

What is the difference between these parses?

How can we decide between these?
A Simple PCFG

Probabilities!

<table>
<thead>
<tr>
<th>Rule</th>
<th>Symbol</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>VP → V NP</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>VP → VP PP</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>PP → P NP</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>P → with</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>V → saw</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>NP → NP PP</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>NP → astronomers</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>NP → stars</td>
<td>0.18</td>
<td></td>
</tr>
<tr>
<td>NP → saw</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>NP → telescope</td>
<td>0.1</td>
<td></td>
</tr>
</tbody>
</table>

Just like n-gram language modeling, PCFGs break the sentence generation process into smaller steps/probabilities.

The probability of a parse is the product of the PCFG rules.

Parsing problems

- Pick a model
  - e.g. CFG, PCFG, ...
- Train (or learn) a model
  - What CFG/PCFG rules should I use?
  - Parameters (e.g. PCFG probabilities)?
- Parsing
  - Determine the parse tree(s) given a sentence
If we have example parsed sentences, how can we learn a set of PCFGs?

- If we have example parsed sentences, how can we learn a set of PCFGs?

Tree Bank

Supervised PCFG Training

<table>
<thead>
<tr>
<th>Rule</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>S \rightarrow NP VP</td>
<td>0.9</td>
</tr>
<tr>
<td>S \rightarrow VP</td>
<td>0.1</td>
</tr>
<tr>
<td>NP \rightarrow Det A N</td>
<td>0.5</td>
</tr>
<tr>
<td>NP \rightarrow Det PP</td>
<td>0.2</td>
</tr>
<tr>
<td>A \rightarrow Adj A</td>
<td>0.4</td>
</tr>
<tr>
<td>PP \rightarrow Prep NP</td>
<td>1.0</td>
</tr>
<tr>
<td>VP \rightarrow V NP</td>
<td>0.7</td>
</tr>
<tr>
<td>VP \rightarrow V PP</td>
<td>0.3</td>
</tr>
</tbody>
</table>

English

Tree Bank

Supervised PCFG Training

<table>
<thead>
<tr>
<th>Rule</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>S \rightarrow NP VP</td>
<td>0.9</td>
</tr>
<tr>
<td>S \rightarrow VP</td>
<td>0.1</td>
</tr>
<tr>
<td>NP \rightarrow Det A N</td>
<td>0.5</td>
</tr>
<tr>
<td>NP \rightarrow Det PP</td>
<td>0.2</td>
</tr>
<tr>
<td>A \rightarrow Adj A</td>
<td>0.4</td>
</tr>
<tr>
<td>PP \rightarrow Prep NP</td>
<td>1.0</td>
</tr>
<tr>
<td>VP \rightarrow V NP</td>
<td>0.7</td>
</tr>
<tr>
<td>VP \rightarrow V PP</td>
<td>0.3</td>
</tr>
</tbody>
</table>

I eat sushi with tuna

What CFG rules occur in this tree?

Estimating PCFG Probabilities

- We can extract the rules from the trees

<table>
<thead>
<tr>
<th>Rule</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>S \rightarrow NP VP</td>
<td>1.0</td>
</tr>
<tr>
<td>NP \rightarrow PP</td>
<td>0.7</td>
</tr>
<tr>
<td>VP \rightarrow V NP</td>
<td>0.3</td>
</tr>
<tr>
<td>V \rightarrow eat</td>
<td>1.0</td>
</tr>
<tr>
<td>NP \rightarrow N PP</td>
<td>0.3</td>
</tr>
<tr>
<td>NP \rightarrow N</td>
<td>1.0</td>
</tr>
<tr>
<td>V \rightarrow saw</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Estimating PCFG Probabilities

- Extract the rules from the trees
- Calculate the probabilities using MLE

\[ p(\alpha \rightarrow \beta | \alpha) = \frac{\text{count}(\alpha \rightarrow \beta)}{\text{count}(\alpha)} \]

\[ P(\alpha \rightarrow \beta | \alpha) = \sum_{\gamma} \frac{\text{count}(\alpha \rightarrow \gamma, \beta)}{\text{count}(\alpha)} \]
Estimating PCFG Probabilities

Occurrences

<table>
<thead>
<tr>
<th>Production</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>S -&gt; NP VP</td>
<td>10</td>
</tr>
<tr>
<td>S -&gt; V NP</td>
<td>3</td>
</tr>
<tr>
<td>S -&gt; VP PP</td>
<td>2</td>
</tr>
<tr>
<td>NP -&gt; N</td>
<td>7</td>
</tr>
<tr>
<td>NP -&gt; N PP</td>
<td>3</td>
</tr>
<tr>
<td>NP -&gt; DT N</td>
<td>6</td>
</tr>
</tbody>
</table>

\[
P( S \rightarrow V \text{ NP}) = ?
\]

\[
P( S \rightarrow V \text{ NP}) = P( S \rightarrow V \text{ NP} | S) = \frac{\text{count}(S \rightarrow V \text{ NP})}{\text{count}(S)} = \frac{3}{15} = \frac{1}{5}
\]

Grammar Equivalence

- **Weak equivalence:** grammars generate same set of strings
  - Grammar 1: NP → DetP N and DetP → a | the
  - Grammar 2: NP → a N | NP → the N

- **Strong equivalence:** grammars have same set of derivation trees
  - With CFGs, possible only with useless rules
  - Grammar 2: NP → a N | NP → the N
  - Grammar 3: NP → a N | NP → the N, DetP → many

Normal Forms

- There are weakly equivalent normal forms (Chomsky Normal Form, Greibach Normal Form)
- A CFG is in Chomsky Normal Form (CNF) if all productions are of one of two forms:
  - A → BC with A, B, C nonterminals
  - A → a, with A a nonterminal and a a terminal
- Every CFG has a weakly equivalent CFG in CNF

CNF Grammar

- S → VP
- VP → VB NP
- VP → VB NP PP
- NP → DT NN
- NP → NN
- NP → NP PP
- PP → IN NP
- DT → the
- IN → with
- VB → film
- VB → trust
- NN → man
- NN → film
- NN → trust

S → VP
VP → VB NP
VP → VP2 PP
VP2 → VB NP
NP → DT NN
NP → NN
NP → NP PP
PP → IN NP
DT → the
IN → with
VB → film
VB → trust
NN → man
NN → film
NN → trust
### Probabilistic Grammar Conversion

<table>
<thead>
<tr>
<th>Original Grammar</th>
<th>Chomsky Normal Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>S → NP VP</td>
</tr>
<tr>
<td>S → Aux NP VP</td>
<td>S → X1 VP</td>
</tr>
<tr>
<td>X1 → Aux NP</td>
<td></td>
</tr>
<tr>
<td>S → VP</td>
<td>S → book</td>
</tr>
<tr>
<td></td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>S → Verb NP</td>
</tr>
<tr>
<td></td>
<td>S → VP NP</td>
</tr>
<tr>
<td>NP → Pronoun</td>
<td>NP → X</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
</tr>
<tr>
<td>NP → Proper-Noun</td>
<td>NP → Houston</td>
</tr>
<tr>
<td></td>
<td>0.16</td>
</tr>
<tr>
<td>NP → Det Nominal</td>
<td>NP → Det Nominal</td>
</tr>
<tr>
<td>Nominal → Noun</td>
<td>Nominal → book</td>
</tr>
<tr>
<td></td>
<td>0.03</td>
</tr>
<tr>
<td>Nominal → Nominal Noun</td>
<td>Nominal → Nominal Noun</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
</tr>
<tr>
<td>Nominal → Nominal PP</td>
<td>Nominal → Nominal PP</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
</tr>
<tr>
<td>VP → Verb</td>
<td>VP → book</td>
</tr>
<tr>
<td></td>
<td>0.1</td>
</tr>
<tr>
<td>VP → VP NP</td>
<td>VP → Verb NP</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
</tr>
<tr>
<td>VP → VP PP</td>
<td>VP → VP PP</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
</tr>
<tr>
<td>PP → Prep NP</td>
<td>PP → Prep NP</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
</tr>
</tbody>
</table>

### Grammar questions

- Can we determine if a sentence is grammatical?
- Given a sentence, can we determine the syntactic structure?
- Can we determine how likely a sentence is to be grammatical? to be an English sentence?
- Can we generate candidate, grammatical sentences?

### Parsing

- Parsing is the field of NLP interested in automatically determining the syntactic structure of a sentence
- Parsing can also be thought of as determining what sentences are “valid” English sentences

- We have a grammar, determine the possible parse tree(s)
- Let’s start with parsing with a CFG (no probabilities)

```
S → NP VP
NP → PP
NP → N PP
VP → V NP
VP → V NP PP
PP → IN N
PP → I
V → eat
N → sushi
N → tuna
IN → with
```

I eat sushi with tuna

approaches? algorithms?
Parsing

- Top-down parsing
  - ends up doing a lot of repeated work
  - doesn’t take into account the words in the sentence until the end!
- Bottom-up parsing
  - constrain based on the words
  - avoids repeated work (dynamic programming)
  - CKY parser

Bottom-up parsing

- start at the bottom (i.e. words) and build the parse tree up from there
- matching right-hand sides and replacing with left-hand sides

Parsing Example

```
S
  / 
VP
  /  
Verb NP
  /  
Det Nominal
  |  
that Noun
  |  
flight

book that flight
```

Top Down Parsing

```
S
  / 
NP VP
  /  
Pronoun
```

```
book that flight
```
Top Down Parsing

$S \\
|-------| \\
| NP   | VP |

Pronoun

X

book

Top Down Parsing

$S \\
|-------| \\
| NP   | VP |

ProperNoun

Top Down Parsing

$S \\
|-------| \\
| NP   | VP |

Det

Nominal

book
Top Down Parsing

S
  NP
  Det Nominal
  book

Top Down Parsing

S
  VP

Top Down Parsing

S
  NP
  VP
  book

Top Down Parsing

S
  VP
Top Down Parsing

S
  └── VP
    └── Verb

Top Down Parsing

S
  └── VP
    └── Verb
      └── book

Top Down Parsing

S
  └── VP
    └── Verb
      └── book

Top Down Parsing

S
  └── VP
    └── Verb
      └── book
        └── that
Top Down Parsing

```
S  
  | 
  V  
      |   
      N  
          | 
          Verb  NP  
              |   
              book  
                  |   
                  Pronoun  
                      |   
                      that  
                          |   
                          ProperNoun
```

Top Down Parsing

```
S  
  | 
  V  
      |   
      N  
          | 
          Verb  NP  
              |   
              book  
                  |   
                  Pronoun
```
Top Down Parsing

S

VP

Verb NP

book ProperNoun

that

Top Down Parsing

S

VP

Verb NP

book Det Nominal

that
Top Down Parsing

S
  \_ VP
  \_ \_ V NP
  \_ \_ Det Nominal
  \_ \_ that Noun
  \_ \_ _ flight

Bottom Up Parsing

Bottom Up Parsing

Bottom Up Parsing

Bottom Up Parsing
Bottom Up Parsing

Nominal
  Nominal
    Nominal
      Noun
        book
        that
        flight

Bottom Up Parsing

Nominal
  Nominal
    Nominal
      Noun
        flight
        that
        book

Nominal
  Nominal
    Nominal
      Noun
        flight
        that
        book

Nominal
  Nominal
    Nominal
      Noun
        flight
        that
        book

Nominal
  Nominal
    Nominal
      Noun
        flight
        that
        book
Bottom Up Parsing

Diagram 1:
- Nominal
  - PP
    - Noun
      - book
    - Det
      - that
  - Nominal
    - flight

Diagram 2:
- Nominal
  - PP
    - Noun
      - book
    - Det
      - that
  - Nominal
    - flight

Diagram 3:
- Verb
  - book
- Det
  - that
- Nominal
  - flight

Diagram 4:
- VP
  - Det
  - Nominal
    - flight

---

[10/6/11]
Bottom Up Parsing

Bottom Up Parsing

Pros/Cons?

- **Top-down:**
  - Only examines parses that could be valid parses (i.e., with an S on top)
  - Doesn’t take into account the actual words!

- **Bottom-up:**
  - Only examines structures that have the actual words as the leaves
  - Examines sub-parses that may not result in a valid parse!
Why is parsing hard?
- Actual grammars are large
- Lots of ambiguity!
  - Most sentences have many parses
  - Some sentences have a lot of parses
  - Even for sentences that are not ambiguous, there is often ambiguity for subtrees (i.e. multiple ways to parse a phrase)

Why is parsing hard?
- I saw the man on the hill with the telescope
- What are some interpretations?

Structural Ambiguity Can Give Exponential Parses
- "I was on the hill that has a telescope when I saw a man."
- "I saw a man who was on the hill that has a telescope on it."
- "I was on the hill when I used the telescope to see a man."

Dynamic Programming Parsing
- To avoid extensive repeated work you must cache intermediate results, specifically found constituents
- Caching (memoizing) is critical to obtaining a polynomial time parsing (recognition) algorithm for CFGs
- Dynamic programming algorithms based on both top-down and bottom-up search can achieve $O(n^3)$ recognition time where $n$ is the length of the input string.
Dynamic Programming Parsing Methods

- **CKY** (Cocke-Kasami-Younger) algorithm based on bottom-up parsing and requires first normalizing the grammar.
- **Earley parser** is based on top-down parsing and does not require normalizing grammar but is more complex.
- These both fall under the general category of chart parsers which retain completed constituents in a chart.

**CKY parser: the chart**

Cell\([i,j]\) contains all constituents covering words \(i\) through \(j\).

Film, the, man, with, trust

All constituents spanning 1-3 or “the man with”

Cell\([i,j]\) contains all constituents covering words \(i\) through \(j\).

How could we figure this out?
CKY parser: the chart

<table>
<thead>
<tr>
<th></th>
<th>Film</th>
<th>the</th>
<th>man</th>
<th>with</th>
<th>trust</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>j=0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cell\([i,j]\) contains all constituents covering words \(i\) through \(j\)

Key: rules are binary and only have two constituents on the right hand side

\[ VP -> VB NP \]
\[ NP -> DT NN \]

See if we can make a new constituent combining any for “the” with any for “man with”
CKY parser: the chart

Film the man with trust

j = 0 1 2 3 4

i = 0

Cell[i,j] contains all constituents covering words i through j

See if we can make a new constituent combining any for “Film” with any for “the man with trust”

CKY parser: the chart

Film the man with trust

j = 0 1 2 3 4

i = 0

Cell[i,j] contains all constituents covering words i through j

See if we can make a new constituent combining any for “Film the” with any for “man with trust”

CKY parser: the chart

Film the man with trust

j = 0 1 2 3 4

i = 0

Cell[i,j] contains all constituents covering words i through j

See if we can make a new constituent combining any for “Film the man with” with any for “trust”

CKY parser: the chart

Film the man with trust

j = 0 1 2 3 4

i = 0

Cell[i,j] contains all constituents covering words i through j

See if we can make a new constituent combining any for “Film the man with” with any for “trust”
CKY parser: the chart

Cell\(i,j\) contains all constituents covering words \(i\) through \(j\)

What if our rules weren't binary?

See if we can make a new constituent combining any for “Film” with any for “the man” with any for “with trust”

What order should we fill the entries in the chart?

What order should we traverse the entries in the chart?
CKY parser: the chart

<table>
<thead>
<tr>
<th></th>
<th>Film</th>
<th>the</th>
<th>man</th>
<th>with</th>
<th>trust</th>
</tr>
</thead>
<tbody>
<tr>
<td>j=0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i=0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i=1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i=2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i=3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i=4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cell(i, j) contains all constituents covering words i through j

From bottom to top, left to right

Top-left along the diagonals moving to the right

---

CKY parser: unary rules

- S -> VP
- VP -> VB NP
- VP -> VP2 PP
- VP2 -> VB NP
- NP -> DT NN
- NP -> NN
- NP -> NP PP
- PP -> IN NP
- DT -> the
- IN -> with
- VB -> film
- VB -> trust
- NN -> man
- NN -> film
- NN -> trust

- Often, we will leave unary rules rather than converting to CNF
- Do these complicate the algorithm?
  - Must check whenever we add a constituent to see if any unary rules apply