Agenda

- HW4, due Thursday
- Questions, comments, concerns?
- Parsing algorithms
  - Left-corner grammar transformation
  - Earley parsing
- Context-sensitive grammar formalisms?

Parse Tree, Derivation

```
NP
PRP
we

VP
VBD
helped

NP
PRP
her

VP
VB

NP
VBD

NN

S
```

Parse Tree, CNF

```
PRP

VP
VBD

NP
we

helped

PRP
her

VP
VB

NP
VBD

NN

S
```

CYK Chart, span 4, midpoint 3

```
<table>
<thead>
<tr>
<th>Span</th>
<th>(NP, 0.015, 1, NN, NP)</th>
<th>(NP, 0.025, 2, NN, NP)</th>
<th>(NP, 0.045, 3, NN, NP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(NP, 0.15, 2, NN, NN)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(NP, 0.15, 3, NN, NN)</td>
<td>(NP, 0.15, 3, NN, NN)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>NN</td>
<td>NN</td>
<td>NN</td>
</tr>
</tbody>
</table>
```

Top-down, Bottom-up, Left-corner

```
S

VP

V

NP

S
```

(Adapted from Mark Johnson)
Top-down, Bottom-up, Left-corner

Intuitive?

Left-corner Parsing

- The left corner of a context-free rule is the first symbol on the right hand side:
  \[ S \rightarrow \text{NP VP}: \text{left corner is NP}. \]
- The left-corner of each production is recognized bottom-up, and everything else is predicted top-down.
Top-down, Bottom-up, Left-corner

- Top-down:
  - Right-recursive grammars require finite state size
  - But left-recursive grammars require unbounded state size
- Left-corner
  - Finite-state size for both left-recursive and right-recursive grammars
  - Only center-embedded structures require unbounded stacks

From [Resnik, 1992]:

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Space required</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Left</td>
</tr>
<tr>
<td>Top-down</td>
<td>O(n)</td>
</tr>
<tr>
<td>Bottom-up</td>
<td>O(1)</td>
</tr>
<tr>
<td>Left-corner!</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

What people do
|     | O(1) | O(n) | O(1) |

Building a Left-corner Parser?

- Perform a left-corner transform on grammar G, then can use a top-down parser
  - because the LC-transform converts left-recursion into right-recursion

Left-corner Grammar Transform

- A → aA-a for all A ∈ V, a ∈ T
- A → A-C for all A ∈ V, C → ε ∈ P
- A→X→βA→B for all A ∈ V, B → Xβ ∈ P
- A→A → ε for all A ∈ V

• After transforming the grammar, do... what?

CYK!

CKY: Analysis

• Since it’s bottom up, CKY populates the table with a lot of “phantom constituents”
  - Spans that are constituents, but cannot really occur in the context in which they are suggested
• Conversion of grammar to CNF adds additional non-terminal nodes
  - Leads to weak equivalence wrt original grammar
  - Additional terminal nodes not (linguistically) meaningful: but can be cleaned up with post processing
• Is there a parsing algorithm for arbitrary CFGs that combines dynamic programming and top-down control?
Earley Parsing Algorithm

- One advantage of top-down over bottom-up is that one never builds constituents that cannot be rooted
- Earley parsing motivation
  - Only want to build categories that can be rooted
  - Use a top-down filter
  - Use a chart parsing approach
- Dynamic programming algorithm (surprise)
  - Allows arbitrary CFGs
  - Fills a chart in a single sweep over the input

Earley Parsing: Chart, States

- Chart is an array of length $N + 1$, where $N =$ number of words
- Chart entries represent states:
  - Completed constituents and their locations
  - In-progress constituents
  - Predicted constituents
- Each state contains three items of information:
  - A grammar rule
  - Information about progress made in completing the sub-tree represented by the rule
  - Span of the sub-tree

Chart Entries: State Examples

- $S \rightarrow \cdot VP \ [0,0]$
  - A $VP$ is predicted at the start of the sentence
- $NP \rightarrow \bullet \ Nominal \ [1,2]$
  - An $NP$ is in progress; the $Det$ goes from 1 to 2
- $VP \rightarrow V \ NP \ [0,3]$
  - A $VP$ has been found starting at 0 and ending at 3

Earley in a nutshell

- Start by predicting $S$
- Step through chart:
  - New predicted states are created from current states
  - New incomplete states are created by advancing existing states as new constituents are discovered
  - States are completed when rules are satisfied
  - Termination: look for $S \rightarrow \alpha \ [0, N]$

Earley Algorithm

```
function EARLEY-PARSE(words, grammar) returns chart
    ENQUEUE($\gamma \rightarrow \cdot S \ [0,0]$, chart[0])
    for i ← from 0 to LENGTH(words) do
        for each state in chart[i] do
            if INCOMPLETE?(state) and NEXT-CAT(state) is not a part of speech then
                PREDICTOR(state)
            elseif INCOMPLETE?(state) and NEXT-CAT(state) is a part of speech then
                SCANNER(state)
            else
                COMPLETER(state)
        end
    end
    return(chart)
```

Earley Algorithm

```
procedure PREDICTOR($A \rightarrow \alpha \cdot B \cdot [i, j]$)
    for each ($B \rightarrow \gamma$) in GRAMMAR.RULES-FOR($B$, grammar) do
        ENQUEUE($B \rightarrow \gamma \cdot [j], chart[j]$)
    end
end

procedure SCANNER($A \rightarrow \alpha \cdot B \cdot [i, j]$)
    if $B \in PARTS-OF-SPEECH(word[j])$ then
        ENQUEUE($B \rightarrow word[j] \cdot [j, j+1], chart[j+1]$)
    end
end

procedure COMPLETER($B \rightarrow \gamma \cdot [i, k]$)
    for each ($A \rightarrow \alpha \cdot B \cdot [i, j]$) in chart[k] do
        ENQUEUE($A \rightarrow \alpha \cdot B \cdot [i, k], chart[k]$)
    end
end
```
Earley Example

- Input: Book that flight
- Desired end state: $S \rightarrow \alpha \cdot [0,3]$
  - Meaning: $S$ spanning from 0 to 3, completed rule

Earley: Chart[0]

<table>
<thead>
<tr>
<th>Rule</th>
<th>Start State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma \rightarrow \bullet S$</td>
<td>[0.0]</td>
<td>Dummy start state</td>
</tr>
<tr>
<td>$S \rightarrow \bullet NP \bullet VP$</td>
<td>[0.0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>$S \rightarrow \bullet Aux \bullet NP \bullet VP$</td>
<td>[0.0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>$S \rightarrow \bullet VP$</td>
<td>[0.0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>$NP \rightarrow \bullet Pronoun$</td>
<td>[0.0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>$NP \rightarrow \bullet Proper-Noun$</td>
<td>[0.0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>$NP \rightarrow \bullet Det \bullet Nominal$</td>
<td>[0.0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>$VP \rightarrow \bullet Verb$</td>
<td>[0.0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>$VP \rightarrow \bullet Verb \bullet NP$</td>
<td>[0.0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>$VP \rightarrow \bullet Verb \bullet NP \bullet PP$</td>
<td>[0.0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>$VP \rightarrow \bullet Verb \bullet PP$</td>
<td>[0.0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>$VP \rightarrow \bullet VP \bullet PP$</td>
<td>[0.0]</td>
<td>Predictor</td>
</tr>
</tbody>
</table>

Note that given a grammar, these entries are the same for all inputs; they can be pre-loaded...

Earley: Chart[1]

<table>
<thead>
<tr>
<th>Rule</th>
<th>Start State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_{12}$</td>
<td>$\bullet Verb \rightarrow book \bullet$</td>
<td>[0.1]</td>
</tr>
<tr>
<td>$S_{13}$</td>
<td>$\bullet VP \rightarrow \bullet Verb$</td>
<td>[0.1]</td>
</tr>
<tr>
<td>$S_{14}$</td>
<td>$\bullet VP \rightarrow \bullet Verb \bullet NP$</td>
<td>[0.1]</td>
</tr>
<tr>
<td>$S_{15}$</td>
<td>$\bullet VP \rightarrow \bullet Verb \bullet NP \bullet PP$</td>
<td>[0.1]</td>
</tr>
<tr>
<td>$S_{16}$</td>
<td>$\bullet VP \rightarrow \bullet Verb \bullet PP$</td>
<td>[0.1]</td>
</tr>
<tr>
<td>$S_{17}$</td>
<td>$\bullet S \rightarrow \bullet VP$</td>
<td>[0.1]</td>
</tr>
<tr>
<td>$S_{18}$</td>
<td>$\bullet VP \rightarrow \bullet VP \bullet PP$</td>
<td>[0.1]</td>
</tr>
<tr>
<td>$S_{19}$</td>
<td>$\bullet NP \rightarrow \bullet Pronoun$</td>
<td>[1.1]</td>
</tr>
<tr>
<td>$S_{20}$</td>
<td>$\bullet NP \rightarrow \bullet Proper-Noun$</td>
<td>[1.1]</td>
</tr>
<tr>
<td>$S_{21}$</td>
<td>$\bullet NP \rightarrow \bullet Det \bullet Nominal$</td>
<td>[1.1]</td>
</tr>
<tr>
<td>$S_{22}$</td>
<td>$\bullet PP \rightarrow \bullet Prep \bullet NP$</td>
<td>[1.1]</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Rule</th>
<th>Start State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_{23}$</td>
<td>$\bullet Det \rightarrow \bullet that \bullet$</td>
<td>[1.2]</td>
</tr>
<tr>
<td>$S_{24}$</td>
<td>$\bullet NP \rightarrow \bullet Det \bullet Nominal$</td>
<td>[1.2]</td>
</tr>
<tr>
<td>$S_{25}$</td>
<td>$\bullet Nominal \rightarrow \bullet Noun$</td>
<td>[2.2]</td>
</tr>
<tr>
<td>$S_{26}$</td>
<td>$\bullet Nominal \rightarrow \bullet Nominal \bullet Noun$</td>
<td>[2.2]</td>
</tr>
<tr>
<td>$S_{27}$</td>
<td>$\bullet Nominal \rightarrow \bullet Nominal \bullet PP$</td>
<td>[2.2]</td>
</tr>
<tr>
<td>$S_{28}$</td>
<td>$\bullet Noun \rightarrow \bullet flight \bullet$</td>
<td>[2.3]</td>
</tr>
<tr>
<td>$S_{29}$</td>
<td>$\bullet Nominal \rightarrow \bullet Noun$</td>
<td>[2.3]</td>
</tr>
<tr>
<td>$S_{30}$</td>
<td>$\bullet NP \rightarrow \bullet Det \bullet Nominal \bullet Noun$</td>
<td>[1.3]</td>
</tr>
<tr>
<td>$S_{31}$</td>
<td>$\bullet Nominal \rightarrow \bullet Nominal \bullet Noun$</td>
<td>[2.3]</td>
</tr>
<tr>
<td>$S_{32}$</td>
<td>$\bullet Nominal \rightarrow \bullet Nominal \bullet PP$</td>
<td>[2.3]</td>
</tr>
<tr>
<td>$S_{33}$</td>
<td>$\bullet VP \rightarrow \bullet Verb \bullet NP$</td>
<td>[0.3]</td>
</tr>
<tr>
<td>$S_{34}$</td>
<td>$\bullet VP \rightarrow \bullet Verb \bullet NP \bullet PP$</td>
<td>[0.3]</td>
</tr>
<tr>
<td>$S_{35}$</td>
<td>$\bullet VP \rightarrow \bullet Prep \bullet NP$</td>
<td>[3.3]</td>
</tr>
<tr>
<td>$S_{36}$</td>
<td>$\bullet S \rightarrow \bullet VP$</td>
<td>[0.3]</td>
</tr>
<tr>
<td>$S_{37}$</td>
<td>$\bullet VP \rightarrow \bullet VP \bullet PP$</td>
<td>[0.3]</td>
</tr>
</tbody>
</table>

Earley: Recovering the Parse

As with CKY, add backpointers...

<table>
<thead>
<tr>
<th>Rule</th>
<th>Start State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{12}$</td>
<td>$\bullet Verb \rightarrow book \bullet$</td>
<td>[0.1]</td>
</tr>
<tr>
<td>$C_{23}$</td>
<td>$\bullet Det \rightarrow that \bullet$</td>
<td>[1.2]</td>
</tr>
<tr>
<td>$C_{28}$</td>
<td>$\bullet Noun \rightarrow flight \bullet$</td>
<td>[2.3]</td>
</tr>
<tr>
<td>$C_{30}$</td>
<td>$\bullet NP \rightarrow Det \bullet Nominal$</td>
<td>[1.3]</td>
</tr>
<tr>
<td>$C_{33}$</td>
<td>$\bullet VP \rightarrow Verb \bullet NP$</td>
<td>[0.3]</td>
</tr>
<tr>
<td>$C_{36}$</td>
<td>$\bullet S \rightarrow VP$</td>
<td>[0.3]</td>
</tr>
</tbody>
</table>

Earley: Efficiency

- For such a simple example, there seems to be a lot of useless stuff...
- Why?
Back to Ambiguity

- Did we solve it?
- No: both CKY and Earley return multiple parse trees…
  - Plus: compact encoding with shared sub-trees
  - Plus: work deriving shared sub-trees is reused
  - Minus: neither algorithm tells us which parse is correct

Ambiguity

- Why don’t humans usually encounter ambiguity?
- How can we improve our models?

Agenda: Summary

- HW4, due Thursday
- Parsing algorithms
  - Earley parsing
  - Left-corner grammar transform
- Next time: context-sensitive grammar formalisms