Basic Parsing with Context-Free Grammars

Some slides adapted from Julia Hirschberg and Dan Jurafsky
Announcements

- To view past videos:
  - [http://globe.cvn.columbia.edu:8080/oncampus.php?c=133ae14752e27fde909fdbd64c06b337](http://globe.cvn.columbia.edu:8080/oncampus.php?c=133ae14752e27fde909fdbd64c06b337)

- Usually available only for 1 week. Right now, available for all previous lectures
Homework Questions?
Evaluation
Syntactic Parsing
Declarative formalisms like CFGs, FSAs define the *legal strings of a language* -- but only tell you ‘this is a legal string of the language X’

Parsing algorithms specify how to recognize the strings of a language and assign each string one (or more) syntactic analyses
Many possible CFGs for English, here is an example (fragment):

- $S \rightarrow NP \ VP$
- $VP \rightarrow V \ NP$
- $NP \rightarrow \text{Det} \ N \mid \text{Adj} \ NP$
- $N \rightarrow \text{boy} \mid \text{girl}$
- $V \rightarrow \text{sees} \mid \text{likes}$
- $\text{Adj} \rightarrow \text{big} \mid \text{small}$
- $\text{DetP} \rightarrow \text{a} \mid \text{the}$

- $^*\text{big the small girl sees a boy}$
- $\text{John likes a girl}$
- $\text{I like a girl}$
- $\text{I sleep}$
- $\text{The old dog the footsteps of the young}$
## Modified CFG

<table>
<thead>
<tr>
<th>S</th>
<th>NP VP</th>
<th>VP</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Aux NP VP</td>
<td>VP -&gt; V PP</td>
<td></td>
</tr>
<tr>
<td>S -&gt; VP</td>
<td>PP -&gt; Prep NP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP</td>
<td>Det Nom</td>
<td>N</td>
<td>old</td>
</tr>
<tr>
<td>NP</td>
<td>PropN</td>
<td>V</td>
<td>dog</td>
</tr>
<tr>
<td>NP -&gt; Pronoun</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nom -&gt; Adj Nom</td>
<td>Aux -&gt; does</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nom</td>
<td>N</td>
<td>Prep -&gt; from</td>
<td>to</td>
</tr>
<tr>
<td>Nom</td>
<td>N Nom</td>
<td>PropN -&gt; Bush</td>
<td>McCain</td>
</tr>
<tr>
<td>Nom</td>
<td>Nom PP</td>
<td>Det -&gt; that</td>
<td>this</td>
</tr>
<tr>
<td>VP</td>
<td>V NP</td>
<td>Adj -&gt; old</td>
<td>green</td>
</tr>
</tbody>
</table>
Parse Tree for ‘The old dog the footsteps of the young’ for Prior CFG
Parsing as a Form of Search

- Searching FSAs
  - Finding the right path through the automaton
  - Search space defined by structure of FSA

- Searching CFGs
  - Finding the right parse tree among all possible parse trees
  - Search space defined by the grammar

- Constraints provided by the input sentence and the automaton or grammar
Top–Down Parser

- Builds from the root S node to the leaves
- Expectation–based
- Common search strategy
  - Top–down, left–to–right, backtracking
  - Try first rule with LHS = S
  - Next expand all constituents in these trees/rules
  - Continue until leaves are POS
  - Backtrack when candidate POS does not match input string
Rule Expansion

“The old dog the footsteps of the young.”
\begin{itemize}
\item Where does backtracking happen?
\item What are the computational disadvantages?
\item What are the advantages?
\end{itemize}
Parser begins with words of input and builds up trees, applying grammar rules whose RHS matches

- Det N V Det N  Prep Det N
- Det Adj N Det N  Prep Det N

The old dog the footsteps of the young.

Parse continues until an S root node reached or no further node expansion possible.
The old dog the footsteps of the young.
Bottom-up parsing

- When does disambiguation occur?
- What are the computational advantages and disadvantages?
What’s right/wrong with….

- **Top–Down parsers** – they never explore illegal parses (e.g. which can’t form an S) -- but waste time on trees that can never match the input
- **Bottom–Up parsers** – they never explore trees inconsistent with input -- but waste time exploring illegal parses (with no S root)
- For both: find a control strategy -- how explore search space efficiently?
  - Pursuing all parses in parallel or backtrack or …?
  - Which rule to apply next?
  - Which node to expand next?
Some Solutions

*Dynamic Programming Approaches – Use a chart to represent partial results*

- CKY Parsing Algorithm
  - Bottom–up
  - Grammar must be in Normal Form
  - The parse tree might not be consistent with linguistic theory
- Early Parsing Algorithm
  - Top–down
  - Expectations about constituents are confirmed by input
  - A POS tag for a word that is not predicted is never added
- Chart Parser
Earley Parsing

- Allows arbitrary CFGs
- Fills a table in a single sweep over the input words
  - Table is length $N+1$; $N$ is number of words
  - Table entries represent
    - Completed constituents and their locations
    - In-progress constituents
    - Predicted constituents
States

- The table-entries are called states and are represented with **dotted-rules**.

\[
\begin{align*}
S & \rightarrow \cdot \text{VP} & \text{A VP is predicted} \\
NP & \rightarrow \text{Det} \cdot \text{Nominal} & \text{An NP is in progress} \\
VP & \rightarrow \text{V NP} \cdot & \text{A VP has been found}
\end{align*}
\]
It would be nice to know where these things are in the input so...

- $S \rightarrow \cdot \ VP \ [0,0]$  
  A VP is predicted at the start of the sentence.

- $NP \rightarrow \ Det \ \cdot \ Nominal \ [1,2]$  
  An NP is in progress; the Det goes from 1 to 2.

- $VP \rightarrow \ V \ NP \ \cdot \ [0,3]$  
  A VP has been found starting at 0 and ending at 3.
As with most dynamic programming approaches, the answer is found by looking in the table in the right place.

In this case, there should be an S state in the final column that spans from 0 to n+1 and is complete.

If that’s the case you’re done.

\[ S \rightarrow \alpha \cdot [0,n+1] \]
Earley Algorithm

- March through chart left-to-right.
- At each step, apply 1 of 3 operators
  - Predictor
    - Create new states representing top-down expectations
  - Scanner
    - Match word predictions (rule with word after dot) to words
  - Completer
    - When a state is complete, see what rules were looking for that completed constituent
Given a state
- With a non-terminal to right of dot (not a part-of-speech category)
- Create a new state for each expansion of the non-terminal
- Place these new states into same chart entry as generated state, beginning and ending where generating state ends.
- So predictor looking at
  - \( S \rightarrow . \ VP \ [0,0] \)
- results in
  - \( VP \rightarrow . \ Verb \ [0,0] \)
  - \( VP \rightarrow . \ Verb \ NP \ [0,0] \)
Scanner

- Given a state
  - With a non-terminal to right of dot that is a part-of-speech category
  - If the next word in the input matches this POS
  - Create a new state with dot moved over the non-terminal
  - So scanner looking at VP -> . Verb NP [0,0]
  - If the next word, “book”, can be a verb, add new state:
    - VP -> Verb . NP [0,1]
  - Add this state to chart entry following current one
  - Note: Earley algorithm uses top-down input to disambiguate POS! Only POS predicted by some state can get added to chart!
Applied to a state when its dot has reached right end of role.
Parser has discovered a category over some span of input.
Find and advance all previous states that were looking for this category
  ◦ copy state, move dot, insert in current chart entry
Given:
  ◦ NP -> Det Nominal . [1,3]
  ◦ VP -> Verb. NP [0,1]
Add
  ◦ VP -> Verb NP . [0,3]
How do we know we are done?

- Find an S state in the final column that spans from 0 to n+1 and is complete.

- If that’s the case you’re done.
  - $S \rightarrow \alpha \cdot [0,n+1]$
More specifically…

1. Predict all the states you can upfront

2. Read a word
   1. Extend states based on matches
   2. Add new predictions
   3. Go to 2

3. Look at N+1 to see if you have a winner
Example

- Book that flight
- We should find… an S from 0 to 3 that is a completed state…
# CFG for Fragment of English

<table>
<thead>
<tr>
<th>S → NP VP</th>
<th>VP → V</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → Aux NP VP</td>
<td>PP -&gt; Prep NP</td>
</tr>
<tr>
<td>NP → Det Nom</td>
<td>N → old</td>
</tr>
<tr>
<td>NP → PropN</td>
<td>V → dog</td>
</tr>
<tr>
<td>Nom -&gt; Adj Nom</td>
<td>Aux → does</td>
</tr>
<tr>
<td>Nom → N</td>
<td>Prep → from</td>
</tr>
<tr>
<td>Nom → N Nom</td>
<td>PropN → Bush</td>
</tr>
<tr>
<td>Nom → Nom PP</td>
<td>Det → that</td>
</tr>
<tr>
<td>VP → V NP</td>
<td>Adj → old</td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>Chart[0]</th>
<th>S0</th>
<th>$\gamma \rightarrow \bullet S$</th>
<th>0,0</th>
<th>Dummy start state</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>$S \rightarrow \bullet NP \ VP$</td>
<td>0,0</td>
<td>Predictor</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>$S \rightarrow \bullet Aux \ NP \ VP$</td>
<td>0,0</td>
<td>Predictor</td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td>$S \rightarrow \bullet VP$</td>
<td>0,0</td>
<td>Predictor</td>
<td></td>
</tr>
<tr>
<td>S4</td>
<td>$NP \rightarrow \bullet Pronoun$</td>
<td>0,0</td>
<td>Predictor</td>
<td></td>
</tr>
<tr>
<td>S5</td>
<td>$NP \rightarrow \bullet Proper-Noun$</td>
<td>0,0</td>
<td>Predictor</td>
<td></td>
</tr>
<tr>
<td>S6</td>
<td>$NP \rightarrow \bullet Det Nominal$</td>
<td>0,0</td>
<td>Predictor</td>
<td></td>
</tr>
<tr>
<td>S7</td>
<td>$VP \rightarrow \bullet Verb$</td>
<td>0,0</td>
<td>Predictor</td>
<td></td>
</tr>
<tr>
<td>S8</td>
<td>$VP \rightarrow \bullet Verb NP$</td>
<td>0,0</td>
<td>Predictor</td>
<td></td>
</tr>
<tr>
<td>S9</td>
<td>$VP \rightarrow \bullet Verb NP PP$</td>
<td>0,0</td>
<td>Predictor</td>
<td></td>
</tr>
<tr>
<td>S10</td>
<td>$VP \rightarrow \bullet Verb PP$</td>
<td>0,0</td>
<td>Predictor</td>
<td></td>
</tr>
<tr>
<td>S11</td>
<td>$VP \rightarrow \bullet VP PP$</td>
<td>0,0</td>
<td>Predictor</td>
<td></td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>Chart[1]</th>
<th>Production Rule</th>
<th>Score</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>S12</td>
<td>$Verb \rightarrow book$</td>
<td>[0,1]</td>
<td>Scanner</td>
</tr>
<tr>
<td>S13</td>
<td>$VP \rightarrow Verb$</td>
<td>[0,1]</td>
<td>Completer</td>
</tr>
<tr>
<td>S14</td>
<td>$VP \rightarrow Verb \cdot NP$</td>
<td>[0,1]</td>
<td>Completer</td>
</tr>
<tr>
<td>S15</td>
<td>$VP \rightarrow Verb \cdot NP PP$</td>
<td>[0,0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>S16</td>
<td>$VP \rightarrow Verb \cdot PP$</td>
<td>[0,0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>S17</td>
<td>$S \rightarrow VP$</td>
<td>[0,1]</td>
<td>Completer</td>
</tr>
<tr>
<td>S18</td>
<td>$VP \rightarrow VP \cdot PP$</td>
<td>[0,1]</td>
<td>Completer</td>
</tr>
<tr>
<td>S19</td>
<td>$NP \rightarrow \bullet Pronoun$</td>
<td>[1,1]</td>
<td>Predictor</td>
</tr>
<tr>
<td>S20</td>
<td>$NP \rightarrow \bullet Proper-Noun$</td>
<td>[1,1]</td>
<td>Predictor</td>
</tr>
<tr>
<td>S21</td>
<td>$NP \rightarrow \bullet Det Nominal$</td>
<td>[1,1]</td>
<td>Predictor</td>
</tr>
<tr>
<td>S22</td>
<td>$PP \rightarrow \bullet Prep NP$</td>
<td>[1,1]</td>
<td>Predictor</td>
</tr>
</tbody>
</table>
## Example

<table>
<thead>
<tr>
<th>Chart[2]</th>
<th>Production</th>
<th>SC</th>
<th>Position</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>S23 Det → that</td>
<td></td>
<td></td>
<td>[1,2]</td>
<td>Scanner</td>
</tr>
<tr>
<td>S24 NP → Det Nominal</td>
<td></td>
<td></td>
<td>[1,2]</td>
<td>Completor</td>
</tr>
<tr>
<td>S25 Nominal → Noun</td>
<td></td>
<td></td>
<td>[2,2]</td>
<td>Predictor</td>
</tr>
<tr>
<td>S26 Nominal → Nominal Noun</td>
<td></td>
<td></td>
<td>[2,2]</td>
<td>Predictor</td>
</tr>
<tr>
<td>S27 Nominal → Nominal PP</td>
<td></td>
<td></td>
<td>[2,2]</td>
<td>Predictor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chart[3]</th>
<th>Production</th>
<th>SC</th>
<th>Position</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>S28 Noun → flight</td>
<td></td>
<td></td>
<td>[2,3]</td>
<td>Scanner</td>
</tr>
<tr>
<td>S29 Nominal → Noun</td>
<td></td>
<td></td>
<td>[2,3]</td>
<td>Completor</td>
</tr>
<tr>
<td>S30 NP → Det Nominal</td>
<td></td>
<td></td>
<td>[1,3]</td>
<td>Completor</td>
</tr>
<tr>
<td>S31 Nominal → Nominal Noun</td>
<td></td>
<td></td>
<td>[2,3]</td>
<td>Completor</td>
</tr>
<tr>
<td>S32 Nominal → Nominal PP</td>
<td></td>
<td></td>
<td>[2,3]</td>
<td>Completor</td>
</tr>
<tr>
<td>S33 VP → Verb NP</td>
<td></td>
<td></td>
<td>[0,3]</td>
<td>Completor</td>
</tr>
<tr>
<td>S34 VP → Verb NP PP</td>
<td></td>
<td></td>
<td>[0,3]</td>
<td>Completor</td>
</tr>
<tr>
<td>S35 PP → Prep NP</td>
<td></td>
<td></td>
<td>[3,3]</td>
<td>Predictor</td>
</tr>
<tr>
<td>S36 S → VP</td>
<td></td>
<td></td>
<td>[0,3]</td>
<td>Completor</td>
</tr>
</tbody>
</table>
What kind of algorithms did we just describe

- Not parsers – recognizers
  - The presence of an S state with the right attributes in the right place indicates a successful recognition.
  - But no parse tree... no parser
  - That’s how we solve (not) an exponential problem in polynomial time
With the addition of a few pointers we have a parser
Augment the “Completer” to point to where we came from.
Augmenting the chart with structural information

<table>
<thead>
<tr>
<th>Chart[1]</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S8</td>
<td>Verb</td>
<td>book</td>
<td>[0,1]</td>
<td>Scanner</td>
</tr>
<tr>
<td>S9</td>
<td>VP</td>
<td>Verb</td>
<td>[0,1]</td>
<td>Completer</td>
</tr>
<tr>
<td>S10</td>
<td>S</td>
<td>VP</td>
<td>[0,1]</td>
<td>Completer</td>
</tr>
<tr>
<td>S11</td>
<td>VP</td>
<td>Verb</td>
<td>NP</td>
<td>[0,1]</td>
</tr>
<tr>
<td>S12</td>
<td>NP</td>
<td>Det</td>
<td>NOMINAL</td>
<td>[1,1]</td>
</tr>
<tr>
<td>S13</td>
<td>NP</td>
<td>Proper-Noun</td>
<td>[1,1]</td>
<td>Predictor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chart[2]</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Det</td>
<td>that</td>
<td>[1,2]</td>
<td>Scanner</td>
<td></td>
</tr>
<tr>
<td>NP</td>
<td>Det</td>
<td>NOMINAL</td>
<td>[1,2]</td>
<td>Completer</td>
</tr>
<tr>
<td>NOMINAL</td>
<td>Noun</td>
<td>[2,2]</td>
<td>Predictor</td>
<td></td>
</tr>
<tr>
<td>NOMINAL</td>
<td>Noun</td>
<td>NOMINAL</td>
<td>[2,2]</td>
<td>Predictor</td>
</tr>
</tbody>
</table>
Retrieving Parse Trees from Chart

- All the possible parses for an input are in the table
- We just need to read off all the backpointers from every complete S in the last column of the table
- Find all the S -> X . [0,N+1]
- Follow the structural traces from the Completer
- Of course, this won’t be polynomial time, since there could be an exponential number of trees
- We can at least represent ambiguity efficiently
Depth-first search will never terminate if grammar is *left recursive* (e.g. NP → NP PP)

\[(A \rightarrow^* \alpha AB, \alpha \rightarrow^* \varepsilon)\]
Solutions:
 ◦ Rewrite the grammar (automatically?) to a *weakly equivalent* one which is not left-recursive
e.g. The man {on the hill with the telescope...}
NP → NP PP *(wanted: Nom plus a sequence of PPs)*
NP → Nom PP
NP → Nom
Nom → Det N
...becomes...
NP → Nom NP'
Nom → Det N
NP’ → PP NP’ *(wanted: a sequence of PPs)*
NP’ → e
• *Not so obvious what these rules mean...*
Harder to detect and eliminate *non–immediate left recursion*

- NP $\rightarrow$ Nom PP
- Nom $\rightarrow$ NP

Fix depth of search explicitly

Rule ordering: non–recursive rules first

- NP $\rightarrow$ Det Nom
- NP $\rightarrow$ NP PP
Another Problem: Structural ambiguity

- Multiple legal structures
  - Attachment (e.g. I saw a man on a hill with a telescope)
  - Coordination (e.g. younger cats and dogs)
  - NP bracketing (e.g. Spanish language teachers)
NP vs. VP Attachment
Solution?
  ◦ Return all possible parses and disambiguate using “other methods”
Summing Up

- Parsing is a search problem which may be implemented with many control strategies
  - Top-Down or Bottom-Up approaches each have problems
    - Combining the two solves some but not all issues
  - Left recursion
  - Syntactic ambiguity
- Next time: Making use of statistical information about syntactic constituents
  - Read Ch 14