Today

- Parsing with CFGs
  - Bottom-up, top-down
  - Ambiguity
  - CKY parsing
  - (Earley)
  - Shallow
Parsing

- Parsing with CFGs refers to the task of assigning proper trees to input strings.
- Proper here means a tree that covers all and only the elements of the input and has an S at the top.
- It doesn’t actually mean that the system can select the correct tree from among all the possible trees.

Parsing

- As with everything of interest, parsing involves a search which involves the making of choices.
- We’ll start with some basic (meaning bad) methods before moving on to the one or two that you need to know.
For Now

- Assume...
  - You have all the words already in some buffer
  - The input isn’t POS tagged
  - We won’t worry about morphological analysis
  - All the words are known

- These are all problematic in various ways, and would have to be addressed in real applications.

Top-Down Search

- Since we’re trying to find trees rooted with an $S$ (Sentences), why not start with the rules that give us an $S$.
- Then we can work our way down from there to the words.
Bottom-Up Parsing

- Of course, we also want trees that cover the input words. So we might also start with trees that link up with the words in the right way.
- Then work your way up from there to larger and larger trees.
Bottom-Up Search

Book that flight

Verb  Det  Noun

Book  that  flight
Bottom-Up Search

Nominal

Verb  Det  Noun

Book  that  flight

Bottom-Up Search

NP

Nominal

Verb  Det  Noun

Book  that  flight
Bottom-Up Search

```
VP
  NP
    Verb  Det  Noun
      Book  that  flight
```

"The old dog the footsteps of the young."

<table>
<thead>
<tr>
<th>Rule</th>
<th>Symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>VP → V</td>
</tr>
<tr>
<td>S → Aux NP VP</td>
<td>VP → V PP</td>
</tr>
<tr>
<td>S → VP</td>
<td>PP → 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 → Adj N</td>
<td>Aux → does</td>
</tr>
<tr>
<td>Nom → N</td>
<td>Prep → from</td>
</tr>
<tr>
<td>Nom → N Nom</td>
<td>PropN → Fido</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>
Top-Down and Bottom-Up

- **Top-down**
  - Only searches for trees that can be answers (i.e. S’s)
  - But also suggests trees that are not consistent with any of the words

- **Bottom-up**
  - Only forms trees consistent with the words
  - But suggests trees that make no sense globally

Control

- Of course, in both cases we left out how to keep track of the search space and how to make choices
  - Which node to try to expand next
  - Which grammar rule to use to expand a node
- One approach is called backtracking.
  - Make a choice, if it works out then fine
  - If not then back up and make a different choice
Problems

- Even with the best filtering, backtracking methods are doomed because of two inter-related problems
  - Ambiguity
  - Shared subproblems

Ambiguity
Example types of ambiguity

- POS
- Attachment
  - PP
  - Coordination (*old dogs and cats*)

Shared Sub-Problems

- No matter what kind of search (top-down or bottom-up or mixed) that we choose.
  - We don’t want to redo work we’ve already done.
  - Unfortunately, naïve backtracking will lead to duplicated work.
Shared Sub-Problems

- Consider
  - A flight from Indianapolis to Houston on TWA

\[
\begin{array}{c}
\text{NP} \\
\text{Det} \quad \text{Nominal} \\
\quad \text{Nominal} \quad \text{PP} \\
\quad \text{Nominal} \quad \text{PP} \\
\quad \text{Nominal} \quad \text{PP} \\
\quad \text{Nominal} \quad \text{PP} \\
\quad \text{Nominal} \\
\end{array}
\]

- Assume a top-down parse making choices among the various Nominal rules.
- In particular, between these two
  - Nominal -> Noun
  - Nominal -> Nominal PP
- Statically choosing the rules in this order leads to the following bad results...
Shared Sub-Problems

\[ NP \]
\[ \overrightarrow{\text{Det} \rightarrow \text{Nominal}} \]
\[ \downarrow \]
\[ a \rightarrow \text{Noun} \]
\[ \downarrow \]
\[ flight... \]
Dynamic Programming

- DP search methods fill tables with partial results and thereby
  - Avoid doing avoidable repeated work
  - Solve exponential problems in polynomial time
  - Efficiently store ambiguous structures with shared sub-parts.
- Two approaches roughly correspond to bottom-up and top-down approaches.
  - CKY
  - Earley

CKY Parsing

- First we’ll limit our grammar to epsilon-free, binary rules (more later)
- Consider the rule $A \rightarrow BC$
  - If there is an $A$ somewhere in the input then there must be a $B$ followed by a $C$ in the input.
  - If the $A$ spans from $i$ to $j$ in the input then there must be some $k$ s.t. $i < k < j$
    - I.e. The $B$ splits from the $C$ someplace.
Problem

- What if your grammar isn’t binary?
  - As in the case of the TreeBank grammar?
- Convert it to binary... any arbitrary CFG can be rewritten into Chomsky-Normal Form automatically.
- What does this mean?
  - The resulting grammar accepts (and rejects) the same set of strings as the original grammar.
  - But the resulting derivations (trees) are different.

Problem

- More specifically, we want our rules to be of the form
  \[ A \rightarrow B C \]
  Or
  \[ A \rightarrow w \]

*That is, rules can expand to either 2 non-terminals or to a single terminal.*
Binarization Intuition

- Eliminate chains of unit productions.
- Introduce new intermediate non-terminals into the grammar that distribute rules with length > 2 over several rules.

So... $S \rightarrow A B C$ turns into

$S \rightarrow X C$ and

$X \rightarrow A B$

Where X is a symbol that doesn’t occur anywhere else in the the grammar.

Sample L1 Grammar

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Lexicon</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow NP VP$</td>
<td>Det → that</td>
</tr>
<tr>
<td>$S \rightarrow Aux NP VP$</td>
<td>Noun → book</td>
</tr>
<tr>
<td>$S \rightarrow VP$</td>
<td>Verb → book</td>
</tr>
<tr>
<td>NP → Pronoun</td>
<td>Pronoun → I</td>
</tr>
<tr>
<td>NP → Proper-Noun</td>
<td>Proper-Noun → Houston</td>
</tr>
<tr>
<td>NP → Det Nominal</td>
<td>Aux → does</td>
</tr>
<tr>
<td>Nominal → Noun</td>
<td>Preposition → from</td>
</tr>
<tr>
<td>Nominal → Nominal Noun</td>
<td></td>
</tr>
<tr>
<td>Nominal → Nominal PP</td>
<td></td>
</tr>
<tr>
<td>VP → Verb</td>
<td></td>
</tr>
<tr>
<td>VP → Verb NP</td>
<td></td>
</tr>
<tr>
<td>VP → Verb NP PP</td>
<td></td>
</tr>
<tr>
<td>VP → Verb PP</td>
<td></td>
</tr>
<tr>
<td>VP → VP PP</td>
<td></td>
</tr>
<tr>
<td>PP → Preposition NP</td>
<td></td>
</tr>
</tbody>
</table>
CNF Conversion

<table>
<thead>
<tr>
<th>$\mathcal{L}_1$ Grammar</th>
<th>$\mathcal{L}_1$ in CNF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow NP \ VP$</td>
<td>$S \rightarrow NP \ VP$</td>
</tr>
<tr>
<td>$S \rightarrow Aux \ NP \ VP$</td>
<td>$S \rightarrow X1 \ VP$</td>
</tr>
<tr>
<td>$S \rightarrow VP$</td>
<td>$S \rightarrow book \mid include \mid prefer$</td>
</tr>
<tr>
<td></td>
<td>$S \rightarrow \text{Verb NP}$</td>
</tr>
<tr>
<td></td>
<td>$S \rightarrow X2 \ PP$</td>
</tr>
<tr>
<td></td>
<td>$S \rightarrow \text{Verb PP}$</td>
</tr>
<tr>
<td></td>
<td>$S \rightarrow \text{VP PP}$</td>
</tr>
<tr>
<td>$NP \rightarrow \text{Pronoun}$</td>
<td>$NP \rightarrow I \mid she \mid me$</td>
</tr>
<tr>
<td>$NP \rightarrow \text{Proper Noun}$</td>
<td>$NP \rightarrow \text{TWA} \mid \text{Houston}$</td>
</tr>
<tr>
<td>$NP \rightarrow \text{Det Nominal}$</td>
<td>$NP \rightarrow \text{Det Nominal}$</td>
</tr>
<tr>
<td>$\text{Nominal} \rightarrow \text{Nominal}$</td>
<td>$\text{Nominal} \rightarrow \text{book} \mid \text{flight} \mid \text{meal} \mid \text{money}$</td>
</tr>
<tr>
<td>$\text{Nominal} \rightarrow \text{Nominal Noun}$</td>
<td>$\text{Nominal} \rightarrow \text{Nominal Noun}$</td>
</tr>
<tr>
<td>$\text{Nominal} \rightarrow \text{Nominal PP}$</td>
<td>$\text{Nominal} \rightarrow \text{Nominal PP}$</td>
</tr>
<tr>
<td>$VP \rightarrow \text{Verb}$</td>
<td>$VP \rightarrow book \mid include \mid prefer$</td>
</tr>
<tr>
<td>$VP \rightarrow \text{Verb NP}$</td>
<td>$VP \rightarrow \text{Verb NP}$</td>
</tr>
<tr>
<td>$VP \rightarrow \text{Verb NP PP}$</td>
<td>$VP \rightarrow \text{X2 PP}$</td>
</tr>
<tr>
<td>$VP \rightarrow \text{Verb PP}$</td>
<td>$X2 \rightarrow \text{Verb NP}$</td>
</tr>
<tr>
<td>$VP \rightarrow \text{VP PP}$</td>
<td>$VP \rightarrow \text{Verb PP}$</td>
</tr>
<tr>
<td>$PP \rightarrow \text{Preposition NP}$</td>
<td>$PP \rightarrow \text{Preposition NP}$</td>
</tr>
</tbody>
</table>

CKY

- So let’s build a table so that an A spanning from $i$ to $j$ in the input is placed in cell $[i,j]$ in the table.
- So a non-terminal spanning an entire string will sit in cell $[0, n]$
  - Hopefully an $S$
- If we build the table bottom-up, we’ll know that the parts of the A must go from $i$ to $k$ and from $k$ to $j$, for some $k$. 

2/2/2017 Speech and Language Processing - Jurafsky and Martin

33
CKY

- Meaning that for a rule like $A \rightarrow B \ C$ we should look for a $B$ in $[i,k]$ and a $C$ in $[k,j]$.
- In other words, if we think there might be an $A$ spanning $i,j$ in the input... AND
  $A \rightarrow B \ C$ is a rule in the grammar THEN
- There must be a $B$ in $[i,k]$ and a $C$ in $[k,j]$ for some $i<k<j$

CKY

- So to fill the table loop over the cell $[i,j]$ values in some systematic way
  - What constraint should we put on that systematic search?
  - For each cell, loop over the appropriate $k$ values to search for things to add.
Note

- We arranged the loops to fill the table a column at a time, from left to right, bottom to top.
  - This assures us that whenever we’re filling a cell, the parts needed to fill it are already in the table (to the left and below)
  - It’s somewhat natural in that it processes the input a left to right a word at a time
    - Known as online

Example

![Example Table and Diagram]
Cell \([i,j]\) contains all constituents (non-terminals) covering words \(i + 1\) through \(j\)
CKY Parser

Book the flight through Houston

S, VP, Verb
Nominal, Noun

None

VP

Det

Nominal, Noun

NP

VP

Nominal, Noun

NP
### CKY Parser

**Book** \(\text{the}\) **flight** through **Houston**

<table>
<thead>
<tr>
<th></th>
<th>S, VP, Verb, Nominal, Noun</th>
<th>S, VP, Nominal, Noun</th>
<th>S VP, X2</th>
<th>NP</th>
<th>VP, X2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Det</td>
<td>None</td>
<td>Det</td>
<td>None</td>
<td>NP</td>
<td>None</td>
</tr>
<tr>
<td>Nominal, Noun</td>
<td>Det</td>
<td>Nominal, Noun</td>
<td>None</td>
<td>Prep</td>
<td></td>
</tr>
</tbody>
</table>

---

### CKY Parser

**Book** **the** **flight** through **Houston**

<table>
<thead>
<tr>
<th></th>
<th>S, VP, Verb, Nominal, Noun</th>
<th>S VP</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Det</td>
<td>None</td>
<td>NP</td>
<td>None</td>
</tr>
<tr>
<td>Nominal, Noun</td>
<td>Det</td>
<td>Nominal, Noun</td>
<td>None</td>
</tr>
</tbody>
</table>
CKY Parser

Book       the        flight    through  Houston

S, VP, Verb,
Nominal, Noun

S, VP,
Nominal

VP

NP

None

Nominal,
Noun

None

Prep

NP

Prep

NP

ProperNoun

CKY Parser

Book       the        flight    through  Houston

S, VP, Verb,
Nominal, Noun

S

NP

None

Nominal,
Noun

None

Prep

NP

ProperNoun
### CKY Parser

<table>
<thead>
<tr>
<th></th>
<th>S, VP, Verb, Nominal, Noun</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>VP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>None</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Det</td>
<td></td>
<td>NP</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Nominal, Noun</td>
<td></td>
<td>None</td>
<td>Nominal</td>
</tr>
<tr>
<td></td>
<td>Prep</td>
<td></td>
<td>PP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NP</td>
<td>ProperNoun</td>
</tr>
</tbody>
</table>

**Book** the **flight** through **Houston**

### CKY Parser

<table>
<thead>
<tr>
<th></th>
<th>S, VP, Verb, Nominal, Noun</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>VP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>None</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Det</td>
<td></td>
<td>NP</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Nominal, Noun</td>
<td></td>
<td>None</td>
<td>Nominal</td>
</tr>
<tr>
<td></td>
<td>Prep</td>
<td></td>
<td>PP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NP</td>
<td>ProperNoun</td>
</tr>
</tbody>
</table>

**Book** the **flight** through **Houston**
Example

Book  the  flight  through  Houston

S, VP, Verb, Nominal, Noun
[0,1]  S, VP, X2
[0,2]  [0,3]  [0,4]  [0,5]

Det  NP  Nominal
[1,2]  [1,3]  [1,4]  [1,5]

Nominal, Noun  Nominal
[2,3]  [2,4]  [2,5]

Prep
[3,4]  [3,5]

NP, Proper-Noun
[4,5]

Filling column 5

Example

Book  the  flight  through  Houston

S, VP, Verb, Nominal, Noun
[0,1]  S, VP, X2
[0,2]  [0,3]  [0,4]  [0,5]

Det  NP  NP
[1,2]  [1,3]  [1,4]  [1,5]

Nominal, Noun  Prep
[2,3]  [2,4]  [3,4]  [3,5]

PP  NP
[3,4]  [4,5]

NP, Proper-Noun
[4,5]

2/2/2017
Speech and Language Processing - Jurafsky and Martin
**Example**

```
<table>
<thead>
<tr>
<th>Book</th>
<th>the</th>
<th>flight</th>
<th>through</th>
<th>Houston</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_VP, VP, N</td>
<td>S_VP, S</td>
<td>S_VP, X2, S3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nominal, Noun</td>
<td>Nominal</td>
<td>Nominal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0,1)</td>
<td>(0,2)</td>
<td>(0,4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Del</td>
<td>NP</td>
<td>NP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1,2)</td>
<td>(1,3)</td>
<td>(1,5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nominal, Noun</td>
<td>Nominal</td>
<td>PP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2,3)</td>
<td>(2,4)</td>
<td>(2,5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prep</td>
<td>PP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(3,4)</td>
<td>(3,5)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP, Proper-Noun</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4,5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**CKY Notes**

- Since it’s bottom up, CKY populates the table with a lot of phantom constituents.
  - Segments that by themselves are constituents but cannot really occur in the context in which they are being suggested.
  - To avoid this we can switch to a top-down control strategy
  - Or we can add some kind of filtering that blocks constituents where they can not happen in a final analysis.
Earley Parsing

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

Back to Ambiguity

- Did we solve it?
Ambiguity

- No...
  - Both CKY and Earley will result in multiple S structures for the $[0,N]$ table entry.
  - They both efficiently store the sub-parts that are shared between multiple parses.
  - And they obviously avoid re-deriving those sub-parts.
  - But neither can tell us which one is right.

Ambiguity

- In most cases, humans don’t notice incidental ambiguity (lexical or syntactic). It is resolved on the fly and never noticed.
  - I ate the spaghetti with chopsticks
  - I ate the spaghetti with meatballs
  - We’ll try to model that with probabilities.
Shallow or Partial Parsing

- Sometimes we don’t need a complete parse tree
  - Information extraction
  - Question answering
- But we would like more than simple POS sequences

Chunking

- Find major but unembedded constituents like NPs, VPs, AdjPs, PPs
  - Most common task: NP chunking of base NPs
  - [NP I] saw [NP the man] on [NP the hill] with [NP a telescope]
  - No attempt to identify full NPs – no recursion, no post-head words
  - No overlapping constituents
  - E.g., if we add PPs or VPs, they may consist only of their heads, e.g. [PP on]
Approaches: FST Chunking

- Use regexps to identify constituents, e.g.
  - NP $\rightarrow$ (DT) NN* NN
  - Find longest matching chunk
  - Hand-built rules
  - No recursion but can cascade FSTs to approximate true CF parser, aggregating larger and larger constituents
  - E.g. Steve Abney's Cass chunker

Figure 13.18

```
The morning flight from Denver has arrived
```
Approaches: ML Chunking

- Require annotated corpus
- Train classifier to classify each element of input in sequence (e.g. IOB Tagging)
  - B (beginning of sequence)
  - I (internal to sequence)
  - O (outside of any sequence)
  - No end-of-chunk coding – it’s implicit
  - Easier to detect the beginning than the end

Book/B_VP that/B_NP flight/I_NP quickly/O

- Train classifier using features from context and current word such as word identity, POS, chunk tags
- Penn Treebank provides a good training corpus, since it has hand-labeled full parses which can be converted to chunks
Chunk Evaluation

- Compare system output with Gold Standard using
  - **Precision**: # correct chunks retrieved/total hypothesized chunks retrieved by system
  - **Recall**: # correct chunks retrieved/total actual chunks in test corpus
  - **F-measure**: (weighted) Harmonic Mean of Precision and Recall
    \[ F_1 = \frac{2PR}{P + R} \]
  - Best: \( F_1 = .96 \) for NP chunking

Summary and Limitations

- Sometimes shallow parsing is enough for task
- Performance quite accurate
- POS tagging accuracy
- Ambiguities and errors in labels of training corpus
**Distribution of Chunks in CONLL Shared Task**

<table>
<thead>
<tr>
<th>Label</th>
<th>Category</th>
<th>Proportion (%)</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>Noun Phrase</td>
<td>51</td>
<td><em>The most frequently cancelled flight</em></td>
</tr>
<tr>
<td>VP</td>
<td>Verb Phrase</td>
<td>20</td>
<td><em>may not arrive</em></td>
</tr>
<tr>
<td>PP</td>
<td>Prepositional Phrase</td>
<td>20</td>
<td><em>to Houston</em></td>
</tr>
<tr>
<td>ADVP</td>
<td>Adverbial Phrase</td>
<td>4</td>
<td><em>earlier</em></td>
</tr>
<tr>
<td>SBAR</td>
<td>Subordinate Clause</td>
<td>2</td>
<td><em>that</em></td>
</tr>
<tr>
<td>ADJP</td>
<td>Adjective Phrase</td>
<td>2</td>
<td><em>late</em></td>
</tr>
</tbody>
</table>

**Summing Up**

- **Parsing as search**: what search strategies to use?
  - Top down
  - Bottom up
  - How to combine?
- **How to parse as little as possible**
  - Dynamic Programming
  - Different policies for ordering states to be processed
- **Shallow Parsing**
Optional: Earley Parsing

States

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

\[
\begin{align*}
S & \rightarrow \cdot \ VP & \text{A VP is predicted} \\
NP & \rightarrow \text{Det} \cdot \text{Nominal} & \text{An NP is in progress} \\
VP & \rightarrow V \ \text{NP} \cdot & \text{A VP has been found}
\end{align*}
\]
States/Locations

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

Earley

- 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 $N$ and is complete. That is,
  - $S \rightarrow \alpha \bullet [0,N]$  
- If that’s the case you’re done.
Earley

- So sweep through the table from 0 to N...
  - New predicted states are created by starting top-down from S
  - New incomplete states are created by advancing existing states as new constituents are discovered
  - New complete states are created in the same way.

More specifically...

1. **Predict** all the states you can upfront
2. Read a word
   - 1. Extend states based on matches
   - 2. Generate new predictions
   - 3. Go to step 2
3. When you’re out of words, look at the chart to see if you have a winner
Filling in the Chart

- 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 POS following dot) to words in input
  - Completer
    - When a state is complete, see what rules were looking for that complete constituent

Predictor

- 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
  - Put predicted states in same chart cell as generating 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 POS category
  - If next word in input matches this POS
  - Create a new state with dot moved past the non-terminal
- E.g., scanner looking at VP -> . Verb NP [0,0]
  - If next word can be a verb, add new state:
    - VP -> Verb . NP [0,1]
  - Add this state to chart entry following current one
- NB: Earley uses top-down input to disambiguate POS -- only POS predicted by some state can be added to chart

Completer

- Given a state
  - Whose dot has reached right end of rule
  - Parser has discovered a constituent over some span of input
  - Find and advance all previous states that are ‘looking for’ this category
  - Copy state, move dot, insert in current chart entry
- E.g., if processing:
  - NP -> Det Nominal . [1,3] and if state expecting an NP like VP -> Verb. NP [0,1] in chart
- Add
  - VP -> Verb NP . [0,3] to same cell of chart
Reaching a Final State

- Find an S state in chart that spans input from 0 to N+1 and is complete
- Declare victory:
  - $S \rightarrow \alpha \cdot [0,N+1]$

Converting from Recognizer to Parser

- Augment the “Completer” to include pointer to each previous (now completed) state
- Read off all the backpointers from every complete S
Example

- Book that flight
- We should find... an S from 0 to 3 that is a completed state...

Chart[0]

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>$\gamma \rightarrow \bullet S$</td>
<td>[0,0]</td>
<td>Dummy start state</td>
</tr>
<tr>
<td>S1</td>
<td>$S \rightarrow \bullet NP VP$</td>
<td>[0,0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>S2</td>
<td>$S \rightarrow \bullet Aux NP VP$</td>
<td>[0,0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>S3</td>
<td>$S \rightarrow \bullet VP$</td>
<td>[0,0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>S4</td>
<td>$NP \rightarrow \bullet Pronoun$</td>
<td>[0,0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>S5</td>
<td>$NP \rightarrow \bullet Proper-Noun$</td>
<td>[0,0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>S6</td>
<td>$NP \rightarrow \bullet Det Nominal$</td>
<td>[0,0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>S7</td>
<td>$VP \rightarrow \bullet Verb$</td>
<td>[0,0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>S8</td>
<td>$VP \rightarrow \bullet Verb NP$</td>
<td>[0,0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>S9</td>
<td>$VP \rightarrow \bullet Verb NP PP$</td>
<td>[0,0]</td>
<td>Predictor</td>
</tr>
<tr>
<td>S10</td>
<td>$VP \rightarrow \bullet VP PP$</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.
### Chart[1]

<table>
<thead>
<tr>
<th>Rule</th>
<th>Action</th>
<th>Progression</th>
</tr>
</thead>
<tbody>
<tr>
<td>S12</td>
<td>Verb → book</td>
<td>[0,1]</td>
</tr>
<tr>
<td>S13</td>
<td>VP → Verb</td>
<td>[0,1]</td>
</tr>
<tr>
<td>S14</td>
<td>VP → Verb • NP</td>
<td>[0,1]</td>
</tr>
<tr>
<td>S15</td>
<td>VP → Verb • NP • PP</td>
<td>[0,1]</td>
</tr>
<tr>
<td>S16</td>
<td>VP → Verb • PP</td>
<td>[0,1]</td>
</tr>
<tr>
<td>S17</td>
<td>S → VP •</td>
<td>[0,1]</td>
</tr>
<tr>
<td>S18</td>
<td>VP → VP • PP</td>
<td>[0,1]</td>
</tr>
<tr>
<td>S19</td>
<td>NP → • Pronoun</td>
<td>[1,1]</td>
</tr>
<tr>
<td>S20</td>
<td>NP → • Proper-Noun</td>
<td>[1,1]</td>
</tr>
<tr>
<td>S21</td>
<td>NP → • Det Nominal</td>
<td>[1,1]</td>
</tr>
<tr>
<td>S22</td>
<td>PP → • Prep NP</td>
<td>[1,1]</td>
</tr>
</tbody>
</table>

### Charts[2] and [3]

<table>
<thead>
<tr>
<th>Rule</th>
<th>Action</th>
<th>Progression</th>
</tr>
</thead>
<tbody>
<tr>
<td>S23</td>
<td>Det → that</td>
<td>[1,2]</td>
</tr>
<tr>
<td>S24</td>
<td>NP → Det • Nominal</td>
<td>[1,2]</td>
</tr>
<tr>
<td>S25</td>
<td>Nominal → • Noun</td>
<td>[2,2]</td>
</tr>
<tr>
<td>S26</td>
<td>Nominal → • Nominal Noun</td>
<td>[2,2]</td>
</tr>
<tr>
<td>S27</td>
<td>Nominal → • Nominal PP</td>
<td>[2,2]</td>
</tr>
<tr>
<td>S28</td>
<td>Noun → fight</td>
<td>[2,3]</td>
</tr>
<tr>
<td>S29</td>
<td>Nominal → Noun •</td>
<td>[2,3]</td>
</tr>
<tr>
<td>S30</td>
<td>NP → Det Nominal •</td>
<td>[1,3]</td>
</tr>
<tr>
<td>S31</td>
<td>Nominal → Nominal • Noun</td>
<td>[2,3]</td>
</tr>
<tr>
<td>S32</td>
<td>Nominal → Nominal • PP</td>
<td>[2,3]</td>
</tr>
<tr>
<td>S33</td>
<td>VP → Verb NP •</td>
<td>[0,3]</td>
</tr>
<tr>
<td>S34</td>
<td>VP → Verb NP • PP</td>
<td>[0,3]</td>
</tr>
<tr>
<td>S35</td>
<td>PP → • Prep NP</td>
<td>[3,3]</td>
</tr>
<tr>
<td>S36</td>
<td>S → VP •</td>
<td>[0,3]</td>
</tr>
<tr>
<td>S37</td>
<td>VP → VP • PP</td>
<td>[0,3]</td>
</tr>
</tbody>
</table>
Efficiency

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

  - It’s predicting things that aren’t consistent with the input
  - That’s the flipside to the CKY problem.

Details

- As with CKY that isn’t a parser until we add the backpointers so that each state knows where it came from.