Parsing

(Context Free Grammars (CFG))

- $N$ a set of non-terminal symbols (or variables)
- $\Sigma$ a set of terminal symbols (disjoint from $N$)
- $R$ a set of productions or rules of the form $A \rightarrow \beta$, where $A$ is a non-terminal and $\beta$ is a string of symbols from $(\Sigma \cup N)^*$
- $S$, a designated non-terminal called the start symbol

(Simple CFG for ATIS English)

Grammar

- $S \rightarrow NP \ VP$
- $S \rightarrow Aux \ NP \ VP$
- $S \rightarrow VP$
- $NP \rightarrow Pronoun$
- $NP \rightarrow Proper-Noun$
- $NP \rightarrow Det \ Nominal$
- $Nominal \rightarrow Noun$
- $Nominal \rightarrow Nominal \ Noun$
- $Nominal \rightarrow Nominal \ PP$
- $VP \rightarrow Verb$
- $VP \rightarrow Verb \ NP$
- $VP \rightarrow VP \ PP$
- $PP \rightarrow Prep \ NP$

Lexicon

- Det $\rightarrow$ the | a | that | this
- Noun $\rightarrow$ book | flight | meal | money
- Verb $\rightarrow$ book | include | prefer
- Pronoun $\rightarrow$ I | he | she | me
- Proper-Noun $\rightarrow$ Houston | NWA
- Aux $\rightarrow$ does
- Prep $\rightarrow$ from | to | on | near | through

Sentence Generation

- Sentences are generated by recursively rewriting the start symbol using the productions until only terminals symbols remain.

Derivation or Parse Tree

Parsing

- Given a string of non-terminals 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.
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
  \  \  \ 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
  \  \ that
Top Down Parsing

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

Bottom Up Parsing

```
book that flight
```

Bottom Up Parsing

```
book that flight
```

Top Down Parsing

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

Nominal
  Noun
    book that flight

Bottom Up Parsing

Nominal
  Noun
    book that flight

Bottom Up Parsing

Nominal
  Noun
    X

Bottom Up Parsing

Nominal
  Noun
    book that flight

Bottom Up Parsing

Nominal
  Noun
    Det
      book that flight

Bottom Up Parsing

Nominal
  PP
    Noun
      Det
        book that flight

Nominal
  NP
    Noun
      Det
        Nominal
          Det
            book that flight
Bottom Up Parsing

VP
  Verb
  book

Det
Nominal
flight

Bottom Up Parsing

S
  VP
  NP

Verb
book
that
Noun
flight

Bottom Up Parsing

S
  VP
  X

Verb
book
that
Noun
flight

Bottom Up Parsing

VP
  Det
  Nominal

Verb
book
that
Noun
flight

Bottom Up Parsing

VP
  PP
  NP

Verb
book
that
Noun
flight

Bottom Up Parsing

VP
  PP
  NP

Verb
book
that
Noun
flight
Bottom Up Parsing

![Diagram of Bottom Up Parsing](image)

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.

Dynamic Programming Parsing
- To avoid extensive repeated work, must cache intermediate results, i.e. completed phrases.
- Caching (memoizing) 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.
- More generally, **chart parsers** retain completed phrases in a chart and can combine top-down and bottom-up search.

CKY
- First grammar must be converted to **Chomsky normal form (CNF)** in which productions must have either exactly 2 non-terminal symbols on the RHS or 1 terminal symbol (lexicon rules).
- Parse bottom-up storing phrases formed from all substrings in a triangular table (chart).
ATIS English Grammar Conversion

Original Grammar

- S → NP VP
- S → Aux NP VP
- S → VP
- NP → Pronoun
- NP → Proper-Noun
- NP → Det Nominal
- Nominal → Noun
- Nominal → Nominal Noun
- Nominal → Nominal PP
- VP → Verb
- VP → Verb NP
- VP → VP PP
- PP → Prep NP

Chomsky Normal Form

- S → NP VP
- S → X1 VP
- X1 → Aux NP
- S → book | include | prefer
- S → Verb NP
- NP → I | he | she | me
- NP → Houston | NWA
- Nominal → book | flight | meal | money
- Nominal → Nominal Noun
- Nominal → Nominal PP
- VP → book | include | prefer
- VP → Verb NP
- VP → VP PP
- PP → Prep NP

CKY Parser

<table>
<thead>
<tr>
<th>Book</th>
<th>the</th>
<th>flight</th>
<th>through Houston</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cell[i,j] contains all constituents (non-terminals) covering words i+1 through j

CKY Parser

<table>
<thead>
<tr>
<th>Book</th>
<th>the</th>
<th>flight</th>
<th>through Houston</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CKY Parser

<table>
<thead>
<tr>
<th>Book</th>
<th>the</th>
<th>flight</th>
<th>through Houston</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Det</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CKY Parser

<table>
<thead>
<tr>
<th>Book</th>
<th>the</th>
<th>flight</th>
<th>through Houston</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Det</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nominal</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Complexity of CKY (recognition)

- There are \( n(n+1)/2 = O(n^2) \) cells.
- Filling each cell requires looking at every possible split point between the two non-terminals needed to introduce a new phrase.
- There are \( O(n) \) possible split points.
- Total time complexity is \( O(n^3) \).

Complexity of CKY (all parses)

- Previous analysis assumes the number of phrase labels in each cell is fixed by the size of the grammar.
- If compute all derivations for each non-terminal, the number of cell entries can expand combinatorially.
- Since the number of parses can be exponential, so is the complexity of finding all parse trees.
Effect of CNF on Parse Trees
- Parse trees are for CNF grammar not the original grammar.
- A post-process can repair the parse tree to return a parse tree for the original grammar.

Syntactic Ambiguity
- Just produces all possible parse trees.
- Does not address the important issue of ambiguity resolution.

Ambiguity

Shallow Parsing
(=Phrase Chunking)

Examples taken from Jurafsky and Martin

Shallow Parsing
(also known as 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 Tagging!
- 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 (Outside) 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  Outside
Evaluating Chunking

- Per token accuracy does not evaluate finding correct full chunks. Instead use:
  \[
  \text{Precision} = \frac{\text{Number of correct chunks found}}{\text{Total number of chunks found}}
  \]
  \[
  \text{Recall} = \frac{\text{Number of correct chunks found}}{\text{Total number of actual chunks}}
  \]
- Take harmonic mean to produce a single evaluation metric called F measure.
  \[
  F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}
  \]
  \[
  F_1 = \frac{1}{\left(\frac{1}{P} + \frac{1}{R}\right)/2} = \frac{2PR}{P + R}
  \]

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 $F_1=92\text{–}94\%$