Grammars and Parsing
CSCI 1460: Computational Linguistics
Lecture 17

Ellie Pavlick
Fall 2022
Topics

- Formal Grammars
- Constituency Parsing with the CKY Algorithm
- Dependency Parsing with the Shift-Reduce Algorithm
Formal Grammar

- Language: (Possibly infinite) set of strings
- Grammar: Set of rewrite rules which define a language
  - Following the rules will generate all the strings that are in the language and none of the strings that are not in the language
  - Allows us to precisely define an (infinite) formal language
- Used also in Programming Languages, Theory of Computation…
Formal Grammar

```
  aa   ab
  aaaaa
  aaaa
  bbb
  cde
  aaa
  a
```
Formal Grammar
Formal Grammar

S → A
A → aa
A → Aaa
Formal Grammar

start symbol

\[ S \rightarrow A \]
\[ A \rightarrow aa \]
\[ A \rightarrow Aaa \]
Formal Grammar

non-terminal symbol

\[ S \rightarrow A \]
\[ A \rightarrow aa \]
\[ A \rightarrow Aaa \]
Formal Grammar

terminal symbol

\[ S \rightarrow A \]
\[ A \rightarrow aa \]
\[ A \rightarrow Aaa \]
Formal Grammar

production rule/rewrite rule

S → A
A → aa
A → Aaa
Formal Grammar

$S \rightarrow A$
$A \rightarrow aa$
$A \rightarrow Aaa$
Formal Grammar

\[
\begin{align*}
S & \rightarrow A \\
A & \rightarrow aa \\
A & \rightarrow Aaa
\end{align*}
\]
Formal Grammar

S → A
A → aa
A → Aaa
Formal Grammar

\[
\begin{align*}
S & \rightarrow A \\
A & \rightarrow aa \\
A & \rightarrow Aaa
\end{align*}
\]
Formal Grammar

\[ S \rightarrow A \]
\[ A \rightarrow aa \]
\[ A \rightarrow Aaa \]

Aaa
Formal Grammar

S → A
A → aa
A → Aaa

Aaa
Formal Grammar

S → A
A → aa
A → Aaa

Aaa
Formal Grammar

S → A
A → aa
A → Aaa

Aaaaa
Formal Grammar

S → A
A → aa
A → Aaa

Aaaaa
Formal Grammar

S → A
A → aa
A → Aaa

aaaaaa

ab
aaaaaa
aaa
aa
aaaa
bb
cde
aa
a
Formal Grammar
Definition

- N: Set of **non-terminal symbols** (can think of these as **variables**)
- Σ: Set of **terminal symbols** (disjoint from N)
- R: Set of **production rules** each of the form $A \rightarrow \beta$
- S: A designated **start symbol** (member of N)
Formal Grammar

Non-Terminal Symbols
\[ N = \{S, A\} \]
Formal Grammar

\[
S \rightarrow A \\
A \rightarrow aa \\
A \rightarrow Aaa
\]

Terminal Symbols
\[\Sigma = \{a\}\]
Formal Grammar

Production Rules

S → A
A → aa
A → Aaa
Formal Grammar

Start Symbol

S → A
A → aa
A → Aaa
Formal Grammar and Automata

- Grammars and Automata are two ways of modeling the same thing
- Grammars are used to *generate* languages
- Automata are used to *recognize* languages
Finite State Automata
Simple Example

“sheep” language: contains “b” followed by at least two “a”s followed by “!”

baa!  
baaa!  
baaaaaaa!

ba!  
baba!  
abaaa!
Finite State Automata

Simple Example

\[
\begin{array}{cccccc}
  b & a & a & a & ! \\
\end{array}
\]

\[
\begin{array}{cccccc}
  \text{q0} & \text{q1} & \text{q2} & \text{q3} & \text{q4} \\
  \text{b} & \text{a} & \text{a} & \text{a} & \text{!} \\
\end{array}
\]
Finite State Automata

Simple Example

start of tape

\[
\begin{array}{cccc}
  b & a & a & a \\
\end{array}
\]

! 

\[
q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_2 \xrightarrow{a} q_3 \xrightarrow{a} q_4
\]
Finite State Automata
Simple Example

$q_0$ b a a a !

$q_0$ -> q1 (b) -> q2 (a) -> q3 (a) -> q4 (!)
Finite State Automata

Simple Example

q0 b a a a !

q0 \(\rightarrow\) q1 b \(\rightarrow\) q2 a \(\rightarrow\) q3 a \(\rightarrow\) q4 !
Finite State Automata
Simple Example

```
b a a a a !
```

![Diagram of a finite state automaton with states q0, q1, q2, q3, and q4, labeled with transitions b→q1, a→q2, a→q3, and !→q4. q1 is labeled as a special state.]
Finite State Automata

Simple Example

q0 \xrightarrow{b} q1 \xrightarrow{a} q2 \xrightarrow{a} q3 \xrightarrow{!} q4
Finite State Automata

Simple Example

$q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_2 \xrightarrow{a} q_3 \xrightarrow{!} q_4$

$b \ a \ a \ a \ a \ !$
Finite State Automata
Simple Example

q0 \rightarrow b \rightarrow q1 \rightarrow a \rightarrow q2 \rightarrow a \rightarrow q3 \rightarrow ! \rightarrow q4
Finite State Automata

Simple Example

q0 \rightarrow \text{b} \rightarrow q1 \rightarrow a \rightarrow q2 \rightarrow a \rightarrow \text{q3} \rightarrow ! \rightarrow q4
Finite State Automata
Simple Example

State Transition Diagram:
- q0 → (b) q1
- q1 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
- q3 → (a) q2
- q2 → (a) q3
Finite State Automata

Simple Example

```
  b a a a !
```

```
q0 -- b --> q1 -- a --> q2 -- a --> q3 -- a --> q4
```

- `b` transitions from `q0` to `q1`.
- `a` transitions from `q1` to `q2`, from `q2` to `q3`, and from `q3` to `q4`.
- `!` transitions from `q3` to `q4`.
Finite State Automata

Simple Example

- Initial state: $q_0$
- Final state: $q_4$
- Transition:
  - $q_0 \xrightarrow{b} q_1$
  - $q_1 \xrightarrow{a} q_2$
  - $q_2 \xrightarrow{a} q_3$
  - $q_3 \xrightarrow{!} q_4$

Tape:

- $b\ a\ a\ a\ !$

End of tape:

- $q_4$ is the final/accept state.
Finite State Automata

Simple Example

q0 \rightarrow b \rightarrow q1 \rightarrow a \rightarrow q2 \rightarrow a \rightarrow q3 \rightarrow ! \rightarrow q4

end of tape

final/accept state
Formal Grammar and Automata

• Regular language:
  • Grammar has the form: $A \rightarrow Ba$ (i.e., one NT symbol, always at the beginning or end)
  • Can be recognized by an finite-state automaton
Formal Grammar and Automata

S → Nom sleeps
Nom → Nom by the tree
Nom → DT NN
DT → {the, a}
NN → {cat, dog, bird, …}
Formal Grammar and Automata

\[ S \rightarrow \text{Nom sleeps} \]
\[ \text{Nom} \rightarrow \text{Nom by the tree} \]
\[ \text{Nom} \rightarrow \text{DT cat} \]
\[ \text{Nom} \rightarrow \text{DT dog} \]
\[ \text{Nom} \rightarrow \text{DT bird} \]
\[ \text{DT} \rightarrow \{\text{the, a}\} \]
Formal Grammar and Automata

S → Nom sleeps
Nom → Nom by the tree
Nom → DT NN
DT → {the, a}
NN → {cat, dog, bird, …}
Formal Grammar and Automata

S → Nom sleeps
Nom → Nom by the tree
Nom → DT NN
DT → {the, a}
NN → {cat, dog, bird, ...}
Formal Grammar and Automata

$$S \rightarrow \text{Nom sleeps}$$
$$\text{Nom} \rightarrow \text{Nom by the tree}$$
$$\text{Nom} \rightarrow \text{DT NN}$$
$$\text{DT} \rightarrow \{\text{the, a}\}$$
$$\text{NN} \rightarrow \{\text{cat, dog, bird, …}\}$$

Nom by the tree sleeps
Formal Grammar and Automata

\[ S \rightarrow \text{Nom sleeps} \]
\[ \text{Nom} \rightarrow \text{Nom by the tree} \]
\[ \text{Nom} \rightarrow \text{DT NN} \]
\[ \text{DT} \rightarrow \{\text{the, a}\} \]
\[ \text{NN} \rightarrow \{\text{cat, dog, bird, ...}\} \]

\text{Nom} by the tree by the tree sleeps
Formal Grammar and Automata

S \rightarrow \text{Nom sleeps}

\text{Nom} \rightarrow \text{Nom by the tree}

\text{Nom} \rightarrow \text{DT NN}

\text{DT} \rightarrow \{\text{the, a}\}

\text{NN} \rightarrow \{\text{cat, dog, bird, …}\}

\text{DT NN by the tree by the tree sleeps}
the cat by the tree by the tree sleeps
the cat sleeps
the cat by the tree sleeps
the cat by the tree by the tree sleeps
...

Formal Grammar and Automata
the cat sleeps
the cats by the tree sleep
the cat by the tree by the tree sleeps
...

Formal Grammar and Automata
Formal Grammar and Automata

• Regular language: doesn’t work when language requires recursion

• Context-Free Languages:
  • Grammars have form $A \rightarrow \alpha$ (where $\alpha$ can be any mix of terminals and non-terminals)
  • Language can be recognized by a pushdown automaton (an automaton that uses a stack to maintain memory)
  • “context free” because rule can be applied to the nonterminal regardless of its context
Formal Grammar and Automata

\[
S \rightarrow \text{NP}_s \ \text{PP} \ \text{VB}_s \\
S \rightarrow \text{NP}_p \ \text{PP} \ \text{VB}_p \\
\text{NP}_s \rightarrow \text{Nom}_s \ \text{PP} \\
\text{Nom}_s \rightarrow \text{DT} \ \text{NN}_s \\
\text{PP} \rightarrow \text{PREP} \ \text{DT} \ \text{NN} \\
\ldots
\]
Chomsky Hierarchy

- recursively enumerable (Turing Machine)
  \[ \alpha A\beta \rightarrow \gamma \]

- context-sensitive (linearly bounded A)
  \[ \alpha A\beta \rightarrow \alpha \gamma \beta \]

- context-free (push-down A)
  \[ A \rightarrow \alpha \]
  \[ A \rightarrow a \]
  \[ A \rightarrow Ba \]

- regular (FSA)
  \[ A \rightarrow a \]
Chomsky Hierarchy

most formal grammars in NLP

enumerable

\[ \alpha A \beta \rightarrow \gamma \]

context-free

\[ \alpha A \beta \rightarrow \alpha \gamma \beta \]

\[ A \rightarrow \alpha \]

regular

\[ A \rightarrow a \]

\[ A \rightarrow B a \]
Chomsky Hierarchy

Swiss-German:
...de Karl d’Maria em Peter de Hans laat hälfte lärne schwüme

English:
...Charles lets Mary help Peter to teach John to Swim

Chomsky Hierarchy

\[ \alpha A \beta \rightarrow \gamma \]
\[ \alpha A \beta \rightarrow \alpha \gamma \beta \]
\[ A \rightarrow \alpha \]
\[ A \rightarrow a \]
\[ A \rightarrow B a \]
Chomsky Hierarchy

Swiss-German:
...de Karl d’Maria em Peter de Hans laat hälfe lärne schwüme

English:
...Charles lets Mary help Peter to teach John to Swim

\[
\begin{align*}
\alpha A \beta &\rightarrow \alpha \gamma \beta \\
\alpha A \beta &\rightarrow \gamma
\end{align*}
\]

\[
\begin{align*}
NN &\rightarrow NN NN VB VB VB VB
\end{align*}
\]

\[
\begin{align*}
A &\rightarrow a \\
A &\rightarrow B a
\end{align*}
\]

context-free

regular
Chomsky Hierarchy

Swiss-German:
...de Karl d’Maria em Peter de Hans laat hälfte lärne schwüme

English:
...Charles lets Mary help Peter to teach John to Swim

regular

\[ \alpha A \beta \rightarrow \alpha \gamma \beta \]

context-free

\[ \alpha A \beta \rightarrow \gamma \]

recursively enumerable

\[ A \rightarrow a \]

\[ A \rightarrow Ba \]
Topics

- Formal Grammars
- **Constituency Parsing with the CKY Algorithm**
- Dependency Parsing with the Shift-Reduce Algorithm
Constituency Grammar

- AKA “Phrase Structure Grammar”
- Organize sentences into phrases that are functionally equivalent (w.r.t, the grammar)
- I.e., phrases can be swapped in and out and maintain grammaticality
- (Note: grammatical ! = meaningful)
Constituency Grammar

The bunny slept soundly under the shade of the umbrella.
The cat played.
The dog with one eye and a green handkerchief jumped and twirled.
Constituency Grammar

The bunny jumped and twirled

The cat slept soundly under the shade of the umbrella

The dog with one eye and a green handkerchief played
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 → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP

Lexicon

Det → that | this | a
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through
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 → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP

Lexicon

Det → that | this | a
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through

book    that    flight
### Grammar

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

### Lexicon

- **Det** → that | this | a
- **Noun** → book | flight | meal | money
- **Verb** → book | include | prefer
- **Pronoun** → I | she | me
- **Proper-Noun** → Houston | NWA
- **Aux** → does
- **Preposition** → from | to | on | near | through
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 → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP

Lexicon:

Det → that | this | a
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through

Diagram:

```
Verb
    Det Nom. Noun
    book that flight
```
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 → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP

Lexicon:
Det → that | this | a
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through

Diagram:
```
  Verb
   \  /
    Det \ Nom.
     \    /
      book that flight
```
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 → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP

Lexicon:

Det → that | this | a
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through
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 → Verb NP PP
VP → Verb PP
VP → VP PP
PP → Preposition NP

Det → that | this | a
Noun → book | flight | meal | money
Verb → book | include | prefer
Pronoun → I | she | me
Proper-Noun → Houston | NWA
Aux → does
Preposition → from | to | on | near | through
CKY Parsing

• Dynamic Programming Algorithm —> breaks problem into sub problems to solve more efficiently
• Basic idea: find a parse for words i through j by combining parses for words i through k with parses for works k through j
CKY Parsing

book 1 the 2 flight 3 through 4 Houston 5
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>book</td>
<td>the</td>
<td>flight</td>
<td>through</td>
<td>Houston</td>
</tr>
</tbody>
</table>

CKY Parsing
CKY Parsing

0 book 1 the 2 flight 3 through 4 Houston 5
CKY Parsing

book 1 the 2 flight 3 through 4 Houston 5
CKY Parsing

book 1 the 2 flight 3 through 4 Houston 5
CKY Parsing
book the flight through Houston
CKY Parsing

0 book 1 the 2 flight 3 through 4 Houston 5
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | this | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | this | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | this | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through

0 book 1 the 2 flight 3 through 4 Houston 5
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through

0  book  1  the  2  flight  3  through  4  Houston
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through

S, VP, Verb, Nominal, Noun
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through
CKY Parsing

```
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through
```
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through

0  book  1  the  2  flight  3  through  4  Houston
CKY Parsing

S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through

0 book 1 the 2 flight 3 through 4 Houston 5

S, VP, Verb, Nominal, Noun

Det
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through
S → NP VP  
S → book | prefer  
S → Verb NP  
S → X2 PP  
S → Verb PP  
S → VP PP  
NP → TWA | Houston  
NP → Det Nominal  
Nominal → book | flight  
Nominal → Nominal PP  
VP → book | prefer  
VP → Verb NP  
VP → X2 PP  
X2 → Verb NP  
VP → Verb PP  
VP → VP PP  
PP → Prep. NP  
Det → that | the | a  
Noun → book | flight  
Verb → book | prefer  
Prop-N → Houston | NWA  
Aux → does  
Prep. → from | to | through
CKY Parsing

S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through

S, VP, Verb, Nominal, Noun
Det
NP
Nom, Noun

book
the
flight
through
Houston

0
1
2
3
4

0
1
2
3
4
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through

0 book 1 the 2 flight 3 through 4 Houston 5
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through
CKY Parsing

book
the
flight
through
Houston

S \rightarrow NP \ VP
S \rightarrow \text{book} \ | \ \text{prefer}
S \rightarrow \text{Verb} \ NP
S \rightarrow \text{X2} \ PP
S \rightarrow \text{Verb} \ PP
S \rightarrow \text{VP} \ PP
NP \rightarrow \text{TWA} \ | \ \text{Houston}
NP \rightarrow \text{Det} \ \text{Nominal}
Nominal \rightarrow \text{book} \ | \ \text{flight}
Nominal \rightarrow \text{Nominal} \ PP
VP \rightarrow \text{book} \ | \ \text{prefer}
VP \rightarrow \text{Verb} \ NP
VP \rightarrow \text{X2} \ PP
X2 \rightarrow \text{Verb} \ NP
VP \rightarrow \text{Verb} \ PP
VP \rightarrow \text{VP} \ PP
PP \rightarrow \text{Prep.} \ NP
Det \rightarrow \text{that} \ | \ \text{the} \ | \ \text{a}
Noun \rightarrow \text{book} \ | \ \text{flight}
Verb \rightarrow \text{book} \ | \ \text{prefer}
Prop-N \rightarrow \text{Houston} \ | \ \text{NWA}
Aux \rightarrow \text{does}
Prep. \rightarrow \text{from} \ | \ \text{to} \ | \ \text{through}
S $\rightarrow$ NP VP
S $\rightarrow$ book | prefer
S $\rightarrow$ Verb NP
S $\rightarrow$ X2 PP
S $\rightarrow$ Verb PP
S $\rightarrow$ VP PP
NP $\rightarrow$ TWA $|$ Houston
NP $\rightarrow$ Det Nominal
Nominal $\rightarrow$ book $|$ flight
Nominal $\rightarrow$ Nominal PP
VP $\rightarrow$ book $|$ prefer
VP $\rightarrow$ Verb NP
VP $\rightarrow$ X2 PP
X2 $\rightarrow$ Verb NP
VP $\rightarrow$ Verb PP
VP $\rightarrow$ VP PP
PP $\rightarrow$ Prep. NP
Det $\rightarrow$ that $|$ the $|$ a
Noun $\rightarrow$ book $|$ flight
Verb $\rightarrow$ book $|$ prefer
Prop-N $\rightarrow$ Houston $|$ NWA
Aux $\rightarrow$ does
Prep. $\rightarrow$ from $|$ to $|$ through

S, VP, Verb, Nominal, Noun

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

S, VP, X2
S, VP
Det
NP
Nom, Noun
Nom
Prep
PP
NP, Prop-N
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through
S → NP VP
S → book | prefer
S → Verb NP
S → X2 PP
S → Verb PP
S → VP PP
NP → TWA | Houston
NP → Det Nominal
Nominal → book | flight
Nominal → Nominal PP
VP → book | prefer
VP → Verb NP
VP → X2 PP
X2 → Verb NP
VP → Verb PP
VP → VP PP
PP → Prep. NP
Det → that | the | a
Noun → book | flight
Verb → book | prefer
Prop-N → Houston | NWA
Aux → does
Prep. → from | to | through

S, VP, Verb,
Nominal, Noun
S, VP, X2
S, VP
Det
NP
Nom, Noun
Nom
Prep
PP
NP, Prop-N
Topics

• Formal Grammars
• Constituency Parsing with the CKY Algorithm
• Dependency Parsing with the Shift-Reduce Algorithm
Dependency Grammar

- Goal is to make explicit the relationships between words (not their substitutability)
- Typically, a verb is at the ROOT
- The core of the sentence is subject-verb-object
- Additional clauses/modifiers hang off of these

I prefer the morning flight through Denver
Dependency Grammar

Figure 14.1  Dependency and constituent analyses for I prefer the morning flight through Denver.
• Dependency parses are represented as a directed graph
  • Vertices (V) and Arcs (A)
• V is roughly the set of words+punctuation
  • In morphologically rich languages, includes stems and affixes too
• Typically constrained to be single-rooted trees
  • This is a simplifying assumption, and means we get some things wrong (like the CFG assumption)
Dependency Parsing

- Transition-Based Parser — specifically “Shift-Reduce” Algorithm
- Basic idea:
  - Progress left to right in the sentence, maintain a stack
  - At each step, either: 1) assign this word to be the head of previous word, 2) assign a previously seen word to be the head of this word, or 3) defer decision to a later step
  - Design about 1, 2, or 3 is made by a supervised classifier
Dependency Parsing

- Three operations, each deals with top 2 elements on the stack
  - LeftArc: set stack[0] as head of stack[1]; remove stack[1] from stack
  - RightArc: set stack[1] as head of stack[0]; remove stack[0] from stack
  - Shift: Push new word onto stack
- I.e., once a word is assigned a head, it is removed
- Preconditions: ROOT can’t have a parent (so can’t apply LeftArc if stack = [*], ROOT, …]
Dependency Parsing

Book me the morning flight
Dependency Parsing

stack = [root]
buffer = [book, me, the, morning, flight]
relations = []
Dependency Parsing

stack = [root]
buffer = [book, me, the, morning, flight]
relations = []

shift!
Dependency Parsing

stack = [root]
buffer = [book, me, the, morning, flight]
relations = []

Step 0: when there are less than 2 things on the stack, always shift.

Book me the morning flight
Dependency Parsing

stack = [root]

buffer = [book, me, the, morning, flight]

relations = []

🔮

Step 0: when there are less than 2 things on the stack, always shift.

book me the morning flight

Step 0: when there are less than 2 things on the stack, always shift.
Dependency Parsing

stack = [book, root]
buffer = [me, the, morning, flight]
relations = []

*Step 0*: when there are less than 2 things on the stack, always shift.

Book me the morning flight
Dependency Parsing

stack = [book, root]
buffer = [me, the, morning, flight]
relations = []
Dependency Parsing

stack = [me, book, root]
buffer = [the, morning, flight]
relations = []

Step 1

Book me the morning flight
Dependency Parsing

stack = [me, book, root]
buffer = [the, morning, flight]
relations = []
Dependency Parsing

stack = [book, root]
buffer = [the, morning, flight]
relations = [book->me]

Step 2: assign book as head of me and remove me from stack
Dependency Parsing

stack = [the, book, root]
buffer = [morning, flight]
relations = [book->me]

Step 3

Book me the morning flight

shift
Dependency Parsing

stack = [morning, the, book, root]
buffer = [flight]
relations = [book->me]

Book me the morning flight

Step 4

shift
Dependency Parsing

stack = [flight, morning, the, book, root]

buffer = []

relations = [book->me]
Dependency Parsing

stack = [flight, the, book, root]
buffer = []
relations = [book->me, flight->morning]

Step 6: assign flight as head of morning, remove morning
left arc!
Dependency Parsing

stack = [flight, book, root]
buffer = []
relations = [book->me, flight->morning, flight->the]

Step 7: assign flight as head of the, remove the
Dependency Parsing

stack = [book, root]
buffer = []
relations = [book->me, flight->morning, flight->the, book->flight]

Step 8: assign book as head of flight, remove flight

right arc!
Dependency Parsing

- \text{stack} = \text{[root]}
- \text{buffer} = \text{[]}
- \text{relations} = \text{[book->me, flight->morning, flight->the, book->flight, root->book]}

Right arc!

Step 9: assign root as head of book, remove book
**Dependency Parsing**

**Figure 14.6** A generic transition-based dependency parser

```latex
\begin{verbatim}
function DEPENDENCYPARSE(words) returns dependency tree

state ← \{[root], [words], []\} ; initial configuration

while state not final
    t ← ORACLE(state) ; choose a transition operator to apply
    state ← APPLY(t, state) ; apply it, creating a new state

return state
\end{verbatim}
```
Dependency Parsing

function DEPENDENCYPARSE(words) returns dependency tree

state ← {{root}, [words], []} ; initial configuration
while state not final
    t ← ORACLE(state) ; choose a transition operator to apply
    state ← APPLY(t, state) ; apply it, creating a new state
return state

ML Classifier

Figure 14.6 A generic transition-based dependency parser
Dependency Parsing
Training the Oracle

- Generate training data deterministically from a treebank (see book)
- Then, the usual ML options are available:
  - Feature-based classifiers (Logistic Regression, Naive Bayes, SVM…)
  - Word-Embedding Features
  - Neural Network Classifiers....
Dependency Parsing
Training the Oracle

Input buffer

Stack

Parser Oracle

Action
LEFTARC
RIGHTARC
SHIFT

Dependency Relations

w3 → w2

ENCODER

w1, w2, w3, w4, w5, w6
All done! Questions?