Announcements

- HW 2 due Friday (new version posted)

- HW 3 is up
Context Free Languages - CFLs

Described in terms of productions (called Backus-Naur Form, or BNF)

- A set of terminals \( T \)
- A set of non-terminals \( N \)
- A start symbol \( S \) (a non-terminal)
- A set of productions
Ex:

- Non-terminals:
  - S → ASA | aB
  - A → B | S
  - B → b | ε

- Terminal
Chomsky Normal Form (CNF)

Each rule in the grammar is either:

• $A \rightarrow BC$ \hspace{1cm} 2 nonterminals
  where neither B or C is the start variable or both are nonterminals

• $A \rightarrow a$ \hspace{1cm} where a is a terminal

• No useless symbols
Why CNF?

Parsing: building those parse trees we saw

In general, there are an exponential number of parse trees for a given input.

Given CNF, can get a polynomial time algorithm to generate a valid parse tree.
Thm: All grammars can be converted to CNF.

Procedure:

- First eliminate useless rules.

- Start from the start state and expand set of "reachable states".

- Start from terminating rules and work backwards.

1. Introduce "dummy" start state
Ex:  \[ S \rightarrow ASA \mid aB \]
\[ A \rightarrow B \mid S \]
\[ B \rightarrow b \mid \varepsilon \]

\[ \exists S, A, B \]
\[ \exists B, A, S \]

\[ S_0 \rightarrow S \]
\[ S \rightarrow ASA \mid aB \]
\[ A \rightarrow B \mid S \]
\[ B \rightarrow b \mid \varepsilon \]
2 Nullable variables: \( B \rightarrow \epsilon \)
(only allowed for \( B = \text{start state in CNF} \) )

Remove all \( \epsilon \) productions:

\[
\begin{align*}
S_0 & \rightarrow S \\
S & \rightarrow ASA \mid aB \\
A & \rightarrow B \\
A & \rightarrow S \\
B & \rightarrow b \\
B & \not\rightarrow b \\
\end{align*}
\]

\[
\begin{align*}
S_0 & \rightarrow S \\
S & \rightarrow ASA \mid aB \mid a \mid B \mid AS \mid SA \\
A & \rightarrow B \\
A & \rightarrow S \\
B & \rightarrow b \\
\end{align*}
\]
(3) Remove unit rules:

\[ A \rightarrow B \]

One idea: if \( A \rightarrow B \) and \( B \Rightarrow w \), remove \( A \rightarrow B \) and replace with \( A \rightarrow w \)

Will work, but 1 problem:

\[ A \rightarrow B \]
\[ B \rightarrow C \mid b \]
\[ C \rightarrow A \mid c \]
Detecting unit pairs:

How? Must have:
\[ X \to Z_i, \ Z_i \to Z_2, \ldots, Z_k \to Y \]

(Since we removed \( \epsilon \) transitions in \( \Theta \))

\[
\text{Compute Unit Pairs (Rules in CFC)}
\]

\[
\text{NewRules} \leftarrow \exists (X \to Y) \text{ in rules } \Theta
\]

\[
\text{do:}
\text{Old Rules} \leftarrow \text{NewRules}
\text{for } (X \to Y) \text{ in New Rules}
\text{for } (Y \to Z) \text{ in New Rules}
\text{New Rules} \leftarrow \text{New Rules} \cup (X \to Z)
\text{while } (\text{NewRules} \neq \text{Old Rules})
\]

\[O(n^2)\]
$A \rightarrow BC$

Now remove all unit rules $A \rightarrow B$.

For any unit pair $(X, Y)$ + rule $Y \rightarrow w$, add $X \rightarrow w$ to the transitions.

Unit Pair: $A \rightarrow B$

only since (non terminal) replacements:

$A \rightarrow X_1, X_1 \rightarrow X_2, X_2 \rightarrow X, \ldots, X_k \rightarrow B$

non terminals
Example:

\[ S_0 \rightarrow S \]
\[ S \rightarrow ASA \mid aB \mid a \mid SA \mid AS \]
\[ A \rightarrow b \]
\[ B \rightarrow b \]

Unit Pairs:

\[ (S_0 \rightarrow S) \quad (A \rightarrow B) \quad (A \rightarrow S) \]

\[ S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS \]
\[ S \rightarrow ASA \mid aB \mid a \mid SA \mid AS \]
\[ A \rightarrow b \mid ASA \mid aB \mid a \mid SA \mid AS \]
\[ B \rightarrow b \]
4) Get rid of “long” right-hand sides.

Recall goal of CNF:

\[ A \rightarrow BC \]  \text{nonterminals}

Not OK: \[ A \rightarrow BCD \]

\[ A \rightarrow bB \]
4a: Create $V_c \rightarrow c$ for every character.

Replace $c$ with $V_c$ everywhere.

Now all rules are either:

$A \rightarrow CDEFG$

or

$V_c \rightarrow c$.
46: \[ A \rightarrow B_1 B_2 B_3 \ldots B_k \]

How to replace with only 2 nonterminals on the right?

\[ A \rightarrow B_1 X_1 \]

\[ X_1 \rightarrow B_2 X_2 \]

\[ X_2 \rightarrow B_3 X_3 \]

\[ \vdots \]

\[ X_{k-1} \rightarrow B_{k-1} B_k \]
Example:

$S_0 \rightarrow ASA | Ub | a | SA | AS$

$S \rightarrow ASA | Ub | a | SA | AS$

$A \rightarrow b | ASA | Ub | a | SA | AS$

$B \rightarrow b$

$U \rightarrow a$

$Z \rightarrow SA$

$Z \rightarrow SA$

Done!
Why CNF?

In general, there are an exponential number of parse trees for a given input.

So how to check quickly?

Even in CNF, might be 2^n possible parse trees:

\[ w_1 \rightarrow w_n \]

Solution: Dynamic Programming
CYK Algorithm (Cocke-Younger-Kasami ’65, ’67)

Given CNF grammar

Given a word \( w = w_1 w_2 w_3 w_4 \ldots w_r \),
we'll look at all possible substrings \( w_i \) \( w_{i+1} \ldots w_j \) and look at how they can be parsed.

We'll build a table from the bottom up.
Ex: \[ S \rightarrow AB \mid BC \]
\[ A \rightarrow BA \]
\[ B \rightarrow CC \]
\[ C \rightarrow AB \]

Test if 'baabe' is in the language.

<table>
<thead>
<tr>
<th>Length 3</th>
<th></th>
<th>EB</th>
<th>EB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length 2</td>
<td>[A, S]</td>
<td>ZB</td>
<td>ZC</td>
</tr>
<tr>
<td>Length 1</td>
<td>[B]</td>
<td>[A, C]</td>
<td>[A, C]</td>
</tr>
</tbody>
</table>

b a a a b a a
Ex (cont)
Running times:
Say we have n rules.
Converting to CNF:

Running CYK: