Announcements

- No office hours today
  check website about tomorrow

- HW is due Friday
Context Free Languages - CFL

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: 

\[
\begin{align*}
S & \rightarrow ASA \mid aB \\
A & \rightarrow B \mid S \\
B & \rightarrow b \mid \varepsilon
\end{align*}
\]
Chomsky Normal Form (CNF)

Each rule in the grammar is either:

- $A \rightarrow BC$
  where neither $B$ or $C$ is the start variable, or both are nonterminals

- $A \rightarrow a$
  where $a$ is a terminal

- No useless symbols

- Only $a$ is at start state
Why CNF?

Parsing: building those parse trees we saw

In general, there are an exponential number of parse trees for a given input. \(\mathcal{O}(n^3)\)

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 + expand set of “reachable states”
- Start from terminating rules + work backwards

Introduce “dummy” start state
So $\rightarrow S$

Ex:

- $S \rightarrow ASA \space\space\space aB$
- $A \rightarrow B \space\space\space S\space\space\space \times\space\space\space C$
- $B \rightarrow b \space\space\space \varepsilon$
- $D \space\space\space \times$
2 Nullable variables: \( B \rightarrow \varepsilon \) (only allowed for \( S_0 \) = start state in CNF)

Remove all \( \varepsilon \) productions:

\[
S_0 \rightarrow S
\]

\[
S \rightarrow ASA | aB | a | SA | AS | S
\]

\[
A \rightarrow B | S
\]

\[
B \rightarrow a | \varepsilon
\]
3. **Remove unit rules:**

\[ A \rightarrow B \]

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

Will work, but 1 problem:

\[
\begin{align*}
A & \rightarrow B \\
B & \rightarrow C \mid b \\
C & \rightarrow A \mid c
\end{align*}
\]

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

How? Must have:

\[ X \rightarrow Z_1, \; Z_1 \rightarrow Z_2, \ldots, Z_k \rightarrow Y \]

(since we removed \( \varepsilon \)-transitions in (2))

```
Compute Unit Pairs (Rules in CFC)

NewRules ← \{ (x → Y) \} in rules \{ \}

do:
    OldRules ← NewRules
    for \( (x → Y) \) in NewRules
        for \( (Y → Z) \) in NewRules
            NewRules ← NewRules ∪ \( (x → Z) \)
    while (NewRules ≠ OldRules)
```
Now remove all unit rules $A \rightarrow B$.

For any unit pair $(X, Y)$, add $X \rightarrow \omega$ to the transitions.
Example: 

\[ S_0 \rightarrow ASA | aB | a | SA | AS \]
\[ S \rightarrow ASA | aB | a | SA | AS \]
\[ A \rightarrow \text{ } \]
\[ B \rightarrow b \]

Pairs: \( (S_0, B), (S_0, S), (A, B), (A, S) \)
4. Get rid of “long” right-hand sides.

Recall goal of CNF:

\[ X \rightarrow YZ \]
\[ \Theta \rightarrow z \]
\[ X \rightarrow aX \]
\[ X \rightarrow XYZ \]

\[ \exists \text{ replace} \]
4a: Create $V_c \rightarrow c$ for every character.

Replace $c$ with $V_c$ everywhere.

Now all rules are either:

$$A \rightarrow CDEF$$

or

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

How to replace with only 2 non-terminals on the right?

\[ A \rightarrow B_1 C \]

\[ C \rightarrow B_2 D \]

\ldots
Ex:

\[ S_0 \rightarrow AX \rightarrow b \rightarrow b \]
\[ S \rightarrow AX \rightarrow B \rightarrow a \rightarrow SA \rightarrow AS \]
\[ A \rightarrow b \rightarrow AX \rightarrow a \rightarrow B \rightarrow a \rightarrow SA \rightarrow AS \]
\[ B \rightarrow b \rightarrow X \rightarrow SA \rightarrow \]

CNF!
Why CNF?

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

So how to check quickly?

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

Solution: dynamic programming!
CYK Algorithm (Cocke-Younger-Kasami ’65, ’67)

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

We'll build a table from the bottom up.
Ex: $S \rightarrow AB \mid BC$

$A \rightarrow BA \mid a$

$B \rightarrow CC \mid b$

$C \rightarrow AB \mid a$

Test if 'baabe' is in the language

$w = w_i \ldots w_n$

$w_{\bar{w}} = w_i \ldots w_j$
Ex (cont)
Running times:
Say we have n rules.
Converting to CNF: $O(n^2)$

Running CYK: $O(n^3)$, $O(n^2)$ space
$O(n^2)$ entries in table
$O(n)$ lookups to fill any 1 entry
Other parsing algorithms

CYK is still pretty slow, especially for large programming languages.

After it was developed, a lot of work was put into figuring out what grammars could have faster algorithms.

Two big (and useful) classes have linear time parsers: LL and LR.

LL(1)  left to right
LR(2)  left versus right-most