Today's learning goals

- Test if a specific string can be generated by a given context-free grammar
- Design a context-free grammar to generate a given language
- Determine if a context-free grammar is ambiguous
- Define push down automata
- Trace the computation of a push down automaton
Birds' eye view

All languages over $\Sigma$

Context-free languages over $\Sigma$

Regular languages over $\Sigma$

Finite languages over $\Sigma$
Designing a CFG

Building a CFG to describe the language

{ abba }

V = {S}
Σ = ? {a, b}
R = { S → abba }
S

What's the set of terminals of this CFG?
A. {a, b}
B. V U S U Σ
C. {S, a, b}
D. {a, b, ε}
E. I don't know.

Can CFGs describe simple sets?
Is every regular language a CFL?

• **Approach 1**: start with an arbitrary DFA $M$, build a CFG that generates $L(M)$.

• **Approach 2**: build CFGs for $\{a\}$, $\{\varepsilon\}$, $\{\}$; then show that the class of CFL is closed under the regular operations (union, concatenation, Kleene star).
Approach 1

Claim: Given any DFA $M$, there is a CFG whose language is $L(M)$.

Construction:
Trace computation using variables to denote state

Given $M = (Q, \Sigma, \delta, q_0, F)$ a DFA, define the CFG

$V = \{ S_i | q_i \text{ is in } Q \}$

$R = \{ S_i \rightarrow aS_j | \delta(q_i, a) = q_j \} \cup \{ S_i \rightarrow \epsilon | q_i \text{ is in } F \}$

$S = S_0$

prove correctness…
Approach 2

- Build CFGs for \{a\}, \{\varepsilon\}, \{\}\; then show that the class of CFL is closed under the regular operations (union, concatenation, Kleene star).

\[
\text{CFG} \quad G \ ::= \ (V, \Sigma, R, S)
\]

\[
\begin{align*}
L & : f a^3 \\
V & : \{S\} \\
R & : S \to a
\end{align*}
\]

\[
\begin{align*}
L & : \{\varepsilon\} \\
V & : \{S\} \\
R & : S \to \varepsilon
\end{align*}
\]

\[
\begin{align*}
L & : \{\}\ \\
V & : \{S\} \\
R & : \{\} \\
\text{no rules}
\end{align*}
\]

Regular Expr.
- \(a \to f a^3\)
- \(\varepsilon \to f \varepsilon^3\)
- \(\emptyset \to \{\}\)

\text{unim concatenation Kleene star}
Approach 2

If $G_1 = (V_1, \Sigma, R_1, S_1)$ and $G_2 = (V_2, \Sigma, R_2, S_2)$ are CFGs and $G_1$ describes $L_1$, $G_2$ describes $L_2$, how can we combine the grammars so we describe $L_1 \cup L_2$?

A. $G = (V_1 \cup V_2, \Sigma, R_1 \cup R_2, S_1 \cup S_2)$

B. $G = (V_1 \times V_2, \Sigma, R_1 \times R_2, (S_1, S_2))$

C. We might not always be able to: the class of CFG describable languages might not be closed under union.

D. I don't know.
Approach 2

If $G_1 = (V_1, \Sigma, R_1, S_1)$ and $G_2 = (V_2, \Sigma, R_2, S_2)$ are CFGs and $G_1$ describes $L_1$, $G_2$ describes $L_2$, how can we combine the grammars so we describe $L_1 \cup L_2$?

$G' = (V, \Sigma, R, S)$

$V = V_1 \cup V_2 \cup \{S\}$

$R = R_1 \cup R_2 \cup \{S \rightarrow S, S_2\}$

(assuming $V_1, V_2$ disjoint)
Designing a CFG

Building a CFG to describe the language

\[ \{ a^n b^n \mid n \geq 0 \} \]

We know this set is not regular!
Designing a CFG

Building a CFG to describe the language

\{ a^n b^n \mid n \geq 0 \} 

One approach:
- what is shortest string in the language?
- how do we go from shorter strings to longer ones?
Designing a CFG

Building a CFG to describe the language

\{ a^n b^n \mid n \geq 0 \}

V = \{ S \}

\Sigma = \{ a,b \}

R =

S

Which rules would complete this CFG?

A. \( S \rightarrow \varepsilon \mid ab \)
B. \( S \rightarrow \varepsilon \mid aS \mid Sb \)
C. \( S \rightarrow \varepsilon \mid aSb \)
D. We need another variable other than S.
E. I don't know.
Designing a CFG

Building a CFG to describe the language

\{ 0^n1^m2^n \mid n,m \geq 0 \}.

Hint: work from the outside in.

Also not a regular set

\[ \begin{align*}
S & \rightarrow \varepsilon | SS2 | T \\
T & \rightarrow \varepsilon | 1T \\
G_\varepsilon (\{S, T, \varepsilon, \leq, R, S\}) & \rightarrow \varepsilon \\
S & \rightarrow^* \varepsilon
\end{align*} \]
Designing a CFG

Building a CFG to describe the language

\{ 0^n1^m2^n | n,m \geq 0 \}.

*Hint: work from the outside in.*

\[ V = \{ S, T \} \]
\[ \Sigma = \{ 0,1,2 \} \]
\[ R = \{ S \rightarrow 0S2 | T | \varepsilon , \quad T \rightarrow 1T | \varepsilon \} \]

S

Also not a regular set
Which of the followings strings is generated by this CFG?

A. E
B. 11
C. 1+1x1
D. ε
E. I don't know.

E \Rightarrow E+E \Rightarrow 1+E \Rightarrow 1+1x1 \Rightarrow 1+1x1

D \Rightarrow \text{olalal... is}
N \Rightarrow D \mid 1N12N1\ldots \mid 13N
Derivations and parsing

$E \rightarrow E + E \mid ExE \mid (E) \mid 1$

Lots of derivations for $1 + 1 \times 1$

$E \rightarrow E + E \rightarrow E + E \times E \rightarrow 1 + E \times E \rightarrow 1 + 1 \times E \rightarrow 1 + 1 \times 1$

$E \rightarrow E \times E \rightarrow E + E \times E \rightarrow 1 + E \times E \rightarrow 1 + 1 \times E \rightarrow 1 + 1 \times 1$

$E \rightarrow E + E \rightarrow 1 + E \rightarrow 1 + E \times E \rightarrow 1 + 1 \times E \rightarrow 1 + 1 \times 1$
Derivations and parsing

E → E+E | ExE | (E) | 1

Lots of derivations for 1+1x1

E → E + E → E + E x E → 1 + E x E → 1 + 1 x E → 1 + 1 x 1

E → E x E → E + E x E → 1 + E x E → 1 + 1 x E → 1 + 1 x 1

E → E + E → 1 + E → 1 + E x E → 1 + 1 x E → 1 + 1 x 1
Lots of derivations:

\[ E \rightarrow E + E \]
\[ E \rightarrow E \times E \]
\[ E \rightarrow 1 + E \]
\[ E \rightarrow 1 + 1 \times E \]

E \[?\] E+E | ExE | (E) | 1

A string is **ambiguously derived** in a CFG if it has more than one leftmost derivation i.e. more than one parsing tree.
Recap: Context-free languages

Context-free grammar

$$G = (V \text{ finite set of variables}, \Sigma \text{ finite set of terminals}, R \text{ finite set of rules}, S \text{ start variable})$$

One-step derivation

$$uAv \rightarrow uwwv$$

where $$u, v, w \in (\Sigma \cup V)^*$$  

$$A \rightarrow w \in R$$

Derivation

$$u \xrightarrow{*} v$$

$$u = v \text{ or } u \rightarrow u_1 \rightarrow \cdots \rightarrow u_k \rightarrow v$$

Language generated by grammar

$$L(G) = \{w \in \Sigma^* | S \xrightarrow{*} w\}$$
An alternative ...

- NFA + stack $\leq_{PDA}$
Pushdown automata

• NFA + stack

At each step
1. **Transition** to new state based on current state, letter read, and top letter of stack.
2. (Possibly) **push or pop** a letter to (or from) top of stack
Pushdown automata

- NFA + stack

Accept a string if there is some sequence of states and some sequence of stack contents which processes the entire input string and ends in an accepting state.
State diagram for PDA

If hand-drawn or in Sipser

State transition labelled $a, b \xrightarrow{c} c$ means "when machine reads an $a$ from the input and the top symbol of the stack is a $b$, it may replace the $b$ with a $c$."

- $a, b \xrightarrow{} \gamma$
- $a$ on input
- $b$ on top of stack
- Pop $b$, don't push anything

- $a, b \xrightarrow{c} c$
- $a$ on the input
- Ignore top stack
- Pushing $c$

- $a, \gamma \xrightarrow{} c$

State diagram for PDA

If hand-drawn or in Sipser

State transition labelled $a, b \xrightarrow{?} c$ means

"when machine reads an a from the input and the top symbol of the stack is a b, it may replace the b with a c."

What edge label would indicate "Read a 0, don't pop anything from stack, don't push anything to the stack"?

A. $0, \varepsilon \xrightarrow{?} \varepsilon$
B. $\varepsilon, 0 \xrightarrow{?} \varepsilon$
C. $\varepsilon, \varepsilon \xrightarrow{?} \varepsilon$
D. $\varepsilon \xrightarrow{?} \varepsilon, 0$
E. I don't know.
Formal definition of PDA

A PDA is a 6-tuple \((Q, \Sigma, \Gamma, \delta, q_0, F)\) where \(Q, \Sigma, \Gamma, F\) are all finite sets and

1. \(Q\) is the set of states

2. \(\Sigma\) is the input alphabet

3. \(\Gamma\) is the stack alphabet

4. \(\delta : Q \times \Sigma \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma)\) is the transition function

5. \(q_0 \in Q\) is the start state

6. \(F \subseteq Q\) is the set of accept states.

\[\text{Thm. } L \text{ generated by CFG } \iff \text{L recognized by PDA}\]