Last Time: Proving Nonregularity

- Focus on computation path through DFA

Idea: if one long string is accepted, then many other strings have to be accepted too.
Pumping Lemma  

If \( A \) is a regular language, then there is a number \( p \) (the pumping length) where, if \( s \) is any string in \( A \) of length at least \( p \), then \( s \) may be divided into three pieces, \( s = x y z \) such that

- \( |y| > 0 \), and
- for each \( i \geq 0 \), \( xy^iz \in A \),
- \( |xy| \leq p \).
**Pumping Lemma**

If $A$ is a regular language, then there is a number $p$ (the pumping length) where, if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into three pieces, $s = x y z$ such that

- $|y| > 0$, and
- for each $i \geq 0$, $x y^i z \in A$,
- $|xy| \leq p$. 

Sipser p. 78 Theorem 1.70
Example

Claim: The set \( \{0^j1^k \mid j,k \geq 0 \text{ and } j \geq k \} \) is not regular.

Proof: ...Consider the string \( s = \ldots \) You must pick \( s \) carefully: we want \(|s| \geq p\) and \( s \) in \( L \). Now prove a contradiction with the statement "\( s \) can be pumped"

Which \( s \) and \( i \) let us complete the proof?

A. \( s = 0^p1^p, \ i=2 \quad \) B. \( s = 0^p1^p, \ i=p \quad \) C. \( s = 0^p1^p, \ i=1 \quad \) D. \( s = 0^p1^p, \ i=0 \quad \) E. I don't know
Regular sets: not the end of the story

• Many **nice / simple / important** sets are not regular
• Limitation of the finite-state automaton model
  • Can't "count"
  • Can only remember finitely far into the past
  • Can't backtrack
  • Must make decisions in "real-time"
• We know computers are more powerful than this model…

*Which conditions should we relax?*
The next model of computation

- **Idea:** allow *some* memory of unbounded size
- **How?**
  - Generalization of regular expressions → **Context-free grammars**
  - Generalization of DFA → **Pushdown Automata**
Context-free grammar  

Informally, a collection of rules used to create strings. CFGs generate *languages*.

\[
S \rightarrow aTb \\
T \rightarrow aT \\
T \rightarrow bT \\
T \rightarrow \epsilon
\]

More formally...
Context-free grammar Sipser Def 2.2, page 102

(V, Σ, R, S)

**Variables:** finite set of (usually upper case) variables, \( V \)

**Terminals:** finite set of alphabet symbols, \( Σ \) \( \quad ) \quad V \cap Σ = \emptyset \)

**Rules / Productions:** finite set of allowed moves, \( R \)

\[ A \rightarrow u \quad A \in V, u \in (V \cup Σ)^* \]

**Start variable:** one variable, \( S \)

\( S \in V \)
Context-free language\textsuperscript{Sipser p. 104}

The language of a CFG \((V, \Sigma, R, S)\) is

\[
\{ w \in \Sigma^* \mid w \text{ can be made from start variable by applying one or more rules} \}
\]
Context-free language

The **language** of a CFG \((V, \Sigma, R, S)\) is

\[ \{ w \in \Sigma^* \mid w \text{ can be made from start variable by applying one or more rules} \} \]

What is the language of the CFG \(\{S\}, \{0\}, R, S\) with rules

\[
S \rightarrow 0S \\
S \rightarrow 0
\]

A. \(\{0, 0S\}\)  
B. \(\{0, 00, 000, \ldots\}\)  
C. \(\{00, 000, \ldots\}\)  
D. \(\{\varepsilon, 0, 00, 000, \ldots\}\)  
E. I don't know.
What is the language of the CFG \(\{S\}, \{0,1\}, R, S\) with rules

\[
\begin{align*}
S & \rightarrow 0S \\
S & \rightarrow 1S \\
S & \rightarrow \varepsilon
\end{align*}
\]

A. \(L(0^*1^*)\) 
B. \(L(0^* \cup 1^*)\) 
C. \(L((0 \cup 1)^*)\) 
D. \(L((0^*1^*)^*)\) 
E. I don't know.
Designing a CFG

Building a CFG to describe the language \{ abba \}

\[ V = \]
\[ \Sigma = \]
\[ R = \]
\[ S = \]

What's the alphabet of this CFG?
A. \{a,b\}
B. \(V \cup S \cup \Sigma\)
C. \{S, a, b\}
D. We get to choose.
E. I don't know.

Can CFGs describe simple sets?
Designing a CFG

Building a CFG to describe the language

\{ abba \}

\[ V = \{ S, T, V, W \} \]

\[ \Sigma = \{ a, b \} \]

\[ R = \{ S \rightarrow aT \quad T \rightarrow bV \quad V \rightarrow bW \quad W \rightarrow a \} \]

S
Designing a CFG – Union

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.
Designing a CFG – Union

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 $L_1 \cup L_2$?
CFL and Regular sets Cor 2.32 Sipser 138

Recall definition of $R$ being a regular expression

1. $R = a$, where $a \in \Sigma$
2. $R = \varepsilon$
3. $R = \emptyset$
4. $R = (R_1 \cup R_2)$, where $R_1, R_2$ are themselves regular expressions
5. $R = (R_1 \circ R_2)$, where $R_1, R_2$ are themselves regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.
CFL and Regular sets

Recall definition of $R$ being a regular expression

1. $R = a$, where $a \in \Sigma$
2. $R = \varepsilon$
3. $R = \emptyset$
4. $R = (R_1 \cup R_2)$, where $R_1, R_2$ are themselves regular expressions
5. $R = (R_1 \circ R_2)$, where $R_1, R_2$ are themselves regular expressions
6. $(R_1^*)$, where $R_1$ is a regular expression.
Claim: Given any DFA $M$, there is a CFG whose language is $L(M)$.

Proof: Trace computation using variables to denote state.
Claim: Given any DFA M, there is a CFG whose language is \( L(M) \).

Proof: Trace computation using variables to denote state

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

\[
V = \{ S_i \mid q_i \text{ is in } Q \}
\]

\[\Sigma\]

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

\[S = S_0\]
Regular languages vs. CFL

Every regular language is context-free.
Designing a CFG

Building a CFG to describe the language

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

$$\{ 0^n 1^m 2^n | n,m \geq 0 \}$$

*Hint: work from the outside in.*

Also not a regular set
Designing a CFG

Building a CFG to describe the language

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

**Hint:** work from the outside in.

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

**Also not a regular set**
CFGs in the wild

V = \{E\}, \Sigma = \{1,+,x,(),\}, R = \{ E \rightarrow E+E | ExE | (E) | 1 \}, S=E

Describing well-formed arithmetic expressions

Which of the followings strings is generated by this CFG?

A. E
B. 11
C. 1+1x1
D. \varepsilon
E. I don't know.
Derivations and parsing

\[ E \rightarrow E+E \mid ExE \mid (E) \mid 1 \]

Lots of derivations for 1+1x1

\[ E \Rightarrow E + E \Rightarrow E + E \times E \Rightarrow 1 + E \times E \Rightarrow 1 + 1 \times E \Rightarrow 1 + 1 \times 1 \]

\[ E \Rightarrow E \times E \Rightarrow E + E \times E \Rightarrow 1 + E \times E \Rightarrow 1 + 1 \times E \Rightarrow 1 + 1 \times 1 \]

\[ E \Rightarrow E + E \Rightarrow 1 + E \Rightarrow 1 + E \times E \Rightarrow 1 + 1 \times E \Rightarrow 1 + 1 \times 1 \]
Derivations and parsing

E → E+E | ExE | (E) | 1

leftmost derivation: replace leftmost variable in each step

Which of these derivations is a leftmost derivation?

E ⇒ E + E ⇒ E + E × E ⇒ 1 + E × E ⇒ 1 + 1 × E ⇒ 1 + 1 × 1

E ⇒ E × E ⇒ E + E × E ⇒ 1 + E × E ⇒ 1 + 1 × E ⇒ 1 + 1 × 1

E ⇒ E + E ⇒ 1 + E ⇒ 1 + E × E ⇒ 1 + 1 × E ⇒ 1 + 1 × 1
A string is **ambiguously derived** in a CFG if it has more than one leftmost derivation i.e. more than one **parse tree**
Recap: Context-free languages

Context-free grammar

\[ G = (V, \Sigma, R, S) \]

One step of a derivation (replaces a variable according to a rule)

\[ uAv \rightarrow uwv \quad \text{where } u, v, w \in (\Sigma \cup V)^* \quad A \rightarrow w \in R \]

Derivation

\[ u \rightarrow^* v \quad u = v \text{ or } u \rightarrow u_1 \rightarrow \cdots \rightarrow u_k \rightarrow v \]

Language generated by grammar

\[ L(G) = \{w \in \Sigma^* | S \rightarrow^* w\} \]