1. Deterministic Finite Automata. Homeworks are p. 78: 2.1.2, 2.1.3, 2.1.7; p. 79: 2.2.1, 2.2.3, 2.3.1, 2.3.2, 2.3.3, 2.4, 2.5, 2.6; p. 81: 2.9, 2.11, 2.12, 2.13; p. 82: 2.14.1, 2.15, 2.16, 2.21, 2.22

- parity of incoming ones:

- even count of zeros after rightmost one:

- Definition: A **deterministic finite automaton** (DFA) is a 5-tuple $M = (Q, \Sigma, \delta, q, F)$, where
  - $Q$ is a **finite** set of **states**
  - $\Sigma$ is a **finite** alphabet of **symbols**
  - $\delta : Q \times \Sigma \rightarrow Q$ is a function (the **transition** function)
  - $q \in Q$ is the **start** state
  - $F \subseteq Q$ is a collection of **accepting** states.

- The transition function is a **program**.

- Note tabular display of the transition function.

- For a set of symbols, $\Sigma$, we let $\Sigma^*$ denote the set of finite strings composed of symbols from $\Sigma$.

- A **language** over $\Sigma$ is a subset of $\Sigma^*$.

- Definition: Let $M = (Q, \Sigma, \delta, q, F)$ be an DFA, and let $w = w_1, w_2, \ldots, w_n \in \Sigma^*$.
  Let $r_0, r_1, \ldots, r_n$ be a sequences of states from $Q$ such that $r_0 = q$ (from the machine definition), and $r_i = \delta(r_{i-1}, w_i)$ for $i = 1, 2, \ldots, n$.
  If $r_n \in F$, we say that $M$ **accepts** $w$.
  If $r_n \notin F$, we say that $M$ **rejects** $w$.

- Definition: Let $M = (Q, \Sigma, \delta, q, F)$ be an DFA.
  Then, $L(M) = \{w \in \Sigma^* : w \text{ is accepted by } M\}$ is the language **accepted** by $M$.

- Definition: A language $A \subseteq \Sigma^*$ is a **regular language** if there exists an DFA $M$ such that $A = L(M)$.
  Note two examples of regular languages at top: “even count of ones” and “even count of zeros after rightmost one.”

- Given DFA $M = (Q, \Sigma, \delta, q, F)$, define $\overline{\delta} : Q \times \Sigma^* \rightarrow Q$ via
  \[
  \overline{\delta}(r, w) = \begin{cases} 
  r, & w = \epsilon \\
  \delta(\overline{\delta}(r, v), a), & w = va, \text{ for } v \in \Sigma^* \text{ and } a \in \Sigma
  \end{cases}
  \]

- In terms of $\overline{\delta}$, $L(M) = \{w \in \Sigma^* : \overline{\delta}(q, w) \in F\}$, where $q$ is the starting state of $M$. 
- Binary string contains 101 as substring:
- Binary string that contains a one in the third position from the right. Use $q_0, q_1, \ldots, q_{11}$ to track last three symbols at any given point in the scan...
- Operations on languages. For languages $A, B \subseteq \Sigma^*$
  - (union) $A \cup B = \{w : w \in A \text{ or } w \in B\}$
  - (concatenation) $AB = \{w : w = uv \text{ for } u \in A, v \in B\}$
  - (Kleen closure, star) $A^* = \{w_1, w_2, \ldots, w_k : k \geq 0, w_i \in A \text{ for } 1 \leq i \leq k\}$
- Kleene closure notation consistent with $\Sigma^*$ notation, where we regard $\Sigma$ as a language containing a finite number of one-character strings.
- Note Kleene closure of any language contains $\epsilon$. So, if $A = \emptyset$, then $A^* = \{\epsilon\}$
- Examples using $A = \{0, 01\}, B = \{1, 10\}$...
- For any language $A \subseteq \Sigma^*$, define
  \begin{align*}
  A^0 &= \{\epsilon\} \\
  A^1 &= A \\
  A^2 &= AA
  \end{align*}
  and so forth. In general $A^k = AA^{k-1}$ and $A^* = \bigcup_{k=0}^{\infty} A^k$.
- Theorem: The set of regular languages is closed under union operations. That is, if $A, B$ are regular languages over the same alphabet, then $A \cup B$ is a regular language.
  Proof: Note: cannot “run” on $A$ then, having failed, run on “B.” A machine gets a single pass over the input string.
  But, can construct a Cartesian product machine...
- Regular languages closed under intersection. Use Cartesian product machine...
- Regular languages closed under complementation: Exchange accepting and non-accepting status for states in accepting DFA for $L$. New machine accept $L^c$.
- Note problems in proving closure under concatenation and Kleene closure...
- In a non-deterministic machine NFA, a machine has several possibilities (including none) at each state
  - The machine may choose to move to one or more states on consuming a given input symbol
  - The machine may choose to move to one or more states without consuming the input symbol
  - The machine may have no choices for a particular input symbol (computation “dies” on that input)
- Binary string contain 101 or 11:
- Trace clones of NFA above for input 010110.
• Binary string with 1 in third place from right:
  DFA had eight states.

• Binary string with zero count = 0 mod 2 or mod 3:
  Definition: A nondeterministic finite automaton (NFA) is a 5-tuple $M = (Q, \Sigma, \delta, q, F)$ with
  - $Q$ is a finite set of states
  - $\Sigma$ is a finite set (alphabet) of symbols
  - $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$
  - $q \in Q$ is the start state
  - $F \subseteq Q$ is a set of accepting states.

• Earlier example

• Definition: Let $M = (Q, \Sigma, \delta, q, F)$ be an NFA and let $w = w_1w_2\ldots w_m \in \Sigma^*$. We say that $M$ accepts $w$ if there exists a state sequence $r_0, r_1, \ldots, r_m \in Q$ with
  - $r_0 = q$
  - For $0 \leq i < m$, $r_{i+1} \in \delta(r_i, y_{i+1})$
  - $r_m \in F$.
  Otherwise, we say $M$ rejects $w$.

• Definition: $M = (Q, \Sigma, \delta, q, F)$ be an NFA. Then the language $L(M)$ accepted by $M$ is $L(M) = \{w \in \Sigma^* : M$ accepts $w\}$.

• Definition: Let $M = (Q, \Sigma, \delta, q, F)$ be an NFA. For $q \in Q$, the $\epsilon$-closure of $q$ is $C_\epsilon(q) = \{r \in Q : r$ can be reached from $q$ by one or more $\epsilon$-transitions$\}$.

• Example: $C_\epsilon(q_1) = \{q_1, q_2\}$:

• Theorem: If $N = (Q, \Sigma, \delta, q, F)$ is an NFA, then there exists a DFA $M$ such that $L(M) = L(N)$. Also, if $M = (Q, \Sigma, \delta, q, F)$ is a DFA, then there exists an NFA $N$ such that $L(N) = L(M)$. 

\[ \begin{array}{c|ccc}
   & 0 & 1 & \epsilon \\
\hline
q_1 & \{q_1\} & \{q_1, q_2\} & \emptyset \\
q_2 & \{q_3\} & \emptyset & \{q_3\} \\
q_3 & \emptyset & \{q_4\} & \emptyset \\
*_{\epsilon}q_4 & \{q_4\} & \emptyset & \emptyset \\
\end{array} \]
Proof: Note DFA to NFA is trivial: convert \( \delta \) by replacing all entries in the tabulation with singleton sets and add an \( \epsilon \) columns with all \( \phi \) entries.

To convert an NFA to a DFA, we will construct a DFA whose state space is the power set of the NFA state space.

Intuition: The constructed DFA, after reading a prefix, say \( w \), of the input will is a subset, which is precisely those states that the NFA could occupy after reading the same prefix...

Here is the construction:
Given \( NFA = (Q, \Sigma, \delta, q, F) \), we construct \( DFA = (Q', \Sigma, \delta', q', F') \), where
- \( Q' = \mathcal{P}(Q) \)
- \( q' = C_\epsilon(q) \)
- \( F' = \{ A \in Q' : A \cap F \neq \emptyset \} \)
- For \( A \in \mathcal{P}(Q) \), define \( \delta'(A, a) = \cup_{r \in A} C_\epsilon(\delta(r, a)) \)

Trace accepting path in NFA as accepting path in DFA — ditto for non-accepting paths
Trace accepting path in DFA as accepting path in NFA — ditto for non-accepting paths.

Theorem: Language \( A \) is regular if and only if there exists an NFA \( N \) with \( A = L(N) \).

Example: Convert to DFA:
• Theorem: Regular language are closed under union. That is, \( A, B \) regular implies \( A \cup B \) regular. (Already have this result from the Cartesian product machine)

Proof: New start state with \( \epsilon \)-transitions to start states of \( A, B \) NFAs. Keep accepting states in both. □

• Theorem: Regular languages are closed under concatenation. That is, \( A, B \) regular implies \( AB \) regular.

Proof: New NFA has start state from \( A \). All accepting states from \( A \) get \( \epsilon \)-transitions to start state of \( B \). Remove accepting status from all such states in \( A \). Keep accepting status for all such states in \( B \). □

• Theorem: Regular languages are closed under Kleene closure.

Proof: Modify NFA \( N \) that accepts language \( L \). New starting/accepting states — to accept \( \epsilon \) string. New start gets \( \epsilon \)-transition to start state of \( M \). All accepting states of \( M \) get \( \epsilon \)-transition to start state of \( M \). □

• Given an alphabet \( \Sigma = \{a_1, a_2, \ldots, a_m\} \), a regular expression over \( \Sigma \) is a constrained string over a related alphabet containing \( \{a_1, a_2, \ldots, a_m\}, (, \cup, *, \epsilon, \phi\} \). Each well-formed regular expression describes a language in \( \Sigma^* \).

• Examples with \( \Sigma = \{0, 1\} \).
  – \( L = \{w : |w| \text{ is even}\} \) via \(((0 \cup 1)(0 \cup 1))^*\)
  – \( L = \{w : |w| \text{ is odd}\} \) via \(((0 \cup 1)(0 \cup 1))^*(0 \cup 1)\)
  – Finite language \( L = \{1011, 0\} \) via \(1011 \cup 0\)
  – \( L = \{w : \text{the first and last symbols of } w \text{ are the same}\} \) via \(0 \cup 1 \cup 0(0 \cup 1)^*0 \cup 0(0 \cup 1)^*1\)

• Recursive definition of a regular expression over alphabet \( \Sigma \).

Base cases: \( \epsilon, \phi, a \in \Sigma \) are regular expressions.

Inductive cases: If \( R_1, R_2 \) are regular expressions, then \( R_1 \cup R_2, R_1R_2 \) are regular expressions. If \( R \) is a regular expression, then \( R^* \) is a regular expression.

• Show how to construct \((0 \cup 1)^*101(0 \cup 1)^*\) from the recursive definition.

• Definition, assuming non-empty alphabet \( \Sigma \)
  – Regular expression \( \epsilon \) represents (describes) the language \( \{\epsilon\}\)
  – Regular expression \( \phi \) represents the language \( \phi \)
- Regular expression $a$, for $a \in \Sigma$, represents the language $\{a\}$
- If regular expressions $R_1, R_2$ represent languages $L_1, L_2$ respectively, then $R_1 \cup R_2$ represents language $L_1 \cup L_2$
- If regular expressions $R_1, R_2$ represent languages $L_1, L_2$ respectively, then $R_1R_2$ represents language $L_1L_2$
- If regular expression $R$ represents language $L$, then $R^*$ represents $L^*$.

- Write $R_1 = R_2$ if $R_1$ and $R_2$ represent the same language.
  
  Example $(0 \cup \epsilon)1^* = 01^* \cup 1^*$.

- List of 16 “identities” that can be used to show regular expressions are equivalent. Some examples:
  - $R\phi = \phi R = \phi$
  - $R\epsilon = \epsilon R = R$
  - $R \cup \phi = \phi \cup R = R$
  - $R \cup R = R$
  - $R_1 \cup R_2 = R_2 \cup R_1$
  - $R_1(R_2 \cup R_3) = R_1R_2 \cup R_1R_3$
  - $\phi^* = \epsilon$
  - $\epsilon^* = \epsilon$

- Theorem: Every regular language expression describes a regular language.
  Proof: (Induction) There are three base cases, corresponding to $R = \epsilon, R = \phi$, or $R = a$, for some $a \in \Sigma$. Here are the corresponding DFAs.

There are three inductive steps: $R_1, R_2 \Rightarrow R_1 \cup R_2$, $R_1, R_2 \Rightarrow R_1R_2$, and $R \Rightarrow R^*$.

But, assuming $R_1, R_2, R$ are regular, we can construct NFAs for $R_1 \cup R_2, R_1R_2,$ and $R^*$ as follows.
• Example: construct a machine for regex \((ab \cup a)^*\):

• Want DFA to regular expression construction. Following recurrence equation will help.

  **Lemma:** Let \(B, C \subseteq \Sigma^*\), with \(\epsilon \not\in B\), and consider the language equation \(L = BL \cup C\). Then, \(L = B^*C\).

  **Proof:** (Side note: for numbers, \(x = 4x + 6 \Rightarrow -3x = 6 \Rightarrow x = -2\.)

  Let \(u \in B^*C\). Want to show \(u \in L\).

  \(u\) is the concatenation of \(k\) strings in \(B\) followed by a string in \(L\), for some \(k \geq 0\).

  Use induction on \(k\).

  **Base case:** \(k = 0 \Rightarrow u \in C \Rightarrow u \in BL \cup C = L\).

  **Inductive case:** \(k > 0 \Rightarrow u = vwc\), with \(v \in B\), \(w\) is the concatenation of \(k - 1\) strings in \(B\), and \(c \in C\).

  Let \(y = wc\), which is the concatenation of \(k - 1\) strings from \(B\) with a string from \(C\).

  By the induction hypothesis, \(y \in L \Rightarrow u = vy \in BL \subseteq BL \cup C = L\). We conclude \(B^*C \subseteq L\).

Now, let \(u \in L\). Want to show \(u \in B^*C\).

Use induction on \(|u|\).

**Base case:** \(|u| = 0 \Rightarrow u = \epsilon \in L = BL \cup C\)

But, \(u = \epsilon \not\in BL\) because \(\epsilon \not\in B\). Therefore \(u \in C \subseteq B^*C\).

**Inductive case:** \(|u| = k > 0\).

  **Case (a)** \(u \in C \Rightarrow u \in B^*C \Rightarrow \) done.

  **Case (b)** \(u \not\in C\)

  Now, since \(u \in L = BL \cup C\), we have \(u \in BL\).

  Then, \(u = bv\), for \(b \in B, v \in L\).

  Then \(\epsilon \not\in B \Rightarrow |b| \geq 1 \Rightarrow |v| < |u| \Rightarrow (induction) v \in B^*C \Rightarrow u = bv \in BB^*C \subseteq B^*C\)

• DFA to regular expression — via an example.

  Given DFA \(M = (Q, \Sigma, \delta, q, F)\), define, for each \(r \in Q\), a language \(L_r\) via \(L_r = \{w \in \Sigma^* : \text{the path starting at state } r \text{ and consuming } w \text{ ends in an accepting state}\}\).

  Note \(L(M) = L_q\), since \(q\) is the start state of \(M\).
We proceed by overkill. That is, we obtain a regular expression for each $L_r$, which includes $L_q = L(M)$.

Example:
We write $L_0$ for $L_{q_0}$ and so forth.

\[
L_0 = aL_0 \cup bL_2 \\
L_1 = aL_0 \cup bL_1 \\
L_2 = \epsilon \cup aL_1 \cup bL_0
\]

Substitute for $L_2$ in equation for $L_0$:

\[
L_0 = aL_0 \cup b(\epsilon \cup aL_1 \cup bL_0) = aL_0 \cup b \cup baL_1 \cup bbL_0 = (a \cup bb)L_0 \cup baL_1 \cup b \\
L_1 = aL_0 \cup bL_1 = bL_1 \cup aL_0 \Rightarrow L_1 = b^*aL_0
\]

Then,

\[
L_0 = (a \cup bb)L_0 \cup bab^*aL_0 \cup b = (a \cup bb \cup bab^*a)L_0 \cup b = (a \cup bb \cup bab^*a)^*b.
\]

General proof involves induction on the number of states.

**Theorem: (Pumping Lemma for Regular Languages).** Let $A$ be a regular language. Then there exists an integer $p \geq 1$ (the pumping length) such that every string in $s \in A$ with $|s| \geq p$ can be written as $s = xyz$, in which the segments $x, y, z$ have the following properties:

- $y \neq \epsilon$
- $|xy| \leq p$
- $xyz \in A$, for $i = 0, 1, 2, 3, \ldots$

Proof: Let $p$ be the number of states in a DFA that accepts $A$. Let $q_0$ be the starting state of the DFA. Consider $s \in A$ with $|s| \geq p$.

Then the accepting trajectory of $s$ involves at least $p$ transitions, which involves at least $p + 1$ states. By the pigeonhole principle, some state, say $q_1$, is repeated within those first $p + 1$ states.
• Over $\Sigma = \{0, 1\}$, $A = \{0^n1^n : n \geq 0\}$ is not regular.
  Proof: Suppose $A$ is regular. Let $p$ be the pumping length. Pump $0^p1^p$ for a contradiction.  

• Over $\Sigma = \{0, 1\}$, $A = \{w \in \Sigma^* : \text{the number of 0s in } w \text{ equals the number of 1s in } w\}$.
  Proof: Suppose $A$ is regular. Let $p$ be the pumping length. Pump $0^p1^p$ for a contradiction.  

• Over $\Sigma = \{0, 1\}$, $A = \{ww : w \in \Sigma^*\}$.
  Proof: Suppose $A$ is regular. Let $p$ be the pumping length. Pump $s = 0^p1^p$ for a contradiction. Note that $s = 0^2p$ doesn’t work. If $y$ is of odd length, then $xz$ is of odd length and is therefore not in $A$.  

• Over $\Sigma = \{0, 1\}$, $A = \{0^m1^n : m > n \geq 0\}$ is not regular.
  Proof: Suppose $A$ is regular. Let $p$ be the pumping length. Pump $s = 0^p1^p$ for a contradiction. Then $xz \in A$ and $xz$ has at least one fewer zeros than $s$, which means the number of zeros is less than or equal to the number of ones, which places $xz \notin A — a contradiction.  

• Over $\Sigma = \{0, 1\}$, $A = \{1^{n^2} : n \geq 0\}$ is not regular.
  Proof: Suppose $A$ is regular. Let $p$ be the pumping length. Pump $s = 1^p^2$.
  
  $|s| = |xyz| = p^2$
  $|xyyz| = |xyz| + |y| = p^2 + |y|$
  $|xy| \leq p \Rightarrow |y| \leq p$
  $y \neq \Rightarrow |y| \geq 1$
  $1 \leq |y| \leq p$
  $p^2 < p^2 + y = |xyyz| \leq p^2 + p < p^2 + 2p + 1 = (p + 1)^2$
  The length of $xyyz$ lies strictly between to adjacent squares: $p^2$ and $(p + 1)^2$ and therefore cannot be a perfect square. We conclude that $xyyz \notin A — a contradiction.  

• Higman’s Theorem (Section 2.10 of the text): For arbitrary $L \in \Sigma^*$,
  $\text{SUBSEQ}(L) = \{x \in \Sigma^* : \exists y \in L, x \text{ is a subsequence of } y\}$ is a regular language.
  Proof: Skip...  


2. Context-free grammars. Homeworks are p. 128: 3.1, 3.2, 3.3; p. 130: 3.6, 3.8.2, 3.8.4, 3.8.5, 3.10

- Definition: A **context-free grammar** is a 4-tuple \((V, \Sigma, R, S)\), where
  - \(V\) is a finite set of **variables**
  - \(\Sigma\) is a finite set of **terminals**
  - \(V \cap \Sigma = \emptyset\)
  - \(S \in V\) is the **start variable**
  - \(R\) is a finite set of **production rules**, each of the form \(A \rightarrow w\), where \(A \in V\) and \(w \in (V \cup \Sigma)^*\).

- Definition: Let \(G = (V, \Sigma, R, S)\) be a context-free grammar. Let \(A \in V\) and \(u, v, w \in (V \cup \Sigma)^*\) such that \(A \rightarrow w \in R\). Then \(uAv\) **derives** \(uwv\) in **one step**, and this process is written as \(uAv \Rightarrow uwv\).

- Definition: Let \(G = (V, \Sigma, R, S)\) be a context-free grammar. Let \(u, v, w \in (V \cup \Sigma)^*\). Then \(u\) **derives** \(v\) in **zero or more steps**, written \(u \Rightarrow^* v\), if
  - \(u = v\), in which case the derivation takes zero steps, or
  - \(\exists k \geq 2\) and a sequence \(u_1, u_2, \ldots, u_k \in (V \cup \Sigma)^*\) such that \(u = u_1 \Rightarrow u_2 \Rightarrow u_3 \Rightarrow \cdots \Rightarrow u_k = v\), requiring \(k - 1\) steps.

- Definition: Let \(G = (V, \Sigma, R, S)\) be a context-free grammar. Then \(L(G)\), the language of \(G\), is \(\{w \in \Sigma^*: S \Rightarrow^* w\}\).

- Example: \(G = \{V = \{A, B, S\}, \Sigma = \{a, b\}, S, R = \{S \rightarrow AB, A \rightarrow a|aA, B \rightarrow b|bB\}\}\)
  - Derive \(aaaabb\)
  - Parse tree, right-most parse, left-most parse, yield
  - \(L(G) = \{a^n b^m : n, m \geq 1\}\).

- Example: Properly nested parenthesis. Let \(a\) be a left parenthesis, \(b\) be a right parenthesis.
  Balance constraints: \(a\)-count = \(b\)-count; unless empty, begins with \(a\), ends with \(b\); scanning from left \(a\)-count \(\geq\) \(b\) count.
  \(G = \{V = \{S\}, \Sigma = \{a, b\}, S, R = \{S \rightarrow \epsilon|SS|aSb\}\}\).
  Note \(L(G)\) is not regular: pump \(a^n b^n\) for \(n\) greater than the pumping length...

- Verifying addition: \(L = \{a^n b^m e^{n+m}\}\).
  Note \(L\) is not regular: pump \(a^p b^p c^{2p}\) ...
  Let \(G = \{V = \{A, B\}, \Sigma = \{a, b, c\}, A, P = \{A \rightarrow aAc|B, B \rightarrow bBc|\epsilon\}\}\).

- Example: regular implies context-free
  \(S \rightarrow 0S|1A\)
  Use grammar:
  \(A \rightarrow 0B|1A\)
  \(B \rightarrow 0S|1C\)
  \(C \rightarrow 0C|1C|\epsilon\)
• Theorem: If \( L \subseteq \Sigma^* \) is a regular language, then \( L \) is a context-free language.

Proof: Let \( L \) be regular. Then there exists DFA \( M = (Q, \Sigma, \delta, q_0, F) \) be the machine that accepts exactly \( L \).

Construct \( G = (V, \Sigma, S, R) \), a context-free grammar as follows.

- \( V = Q \)
- \( S = q_0 \)
- \( R = \{ p \rightarrow aq : ((p, a), q) \in \delta \} \cup \{ q \rightarrow \epsilon : q \in F \} \)

Need to show \( L(G) = L(M) \)

Suppose \( s = s_1s_2 \ldots s_m \in L(M) \).

Then there exists an accepting path in \( M \):

\( q_0 \rightarrow s_1 q_1 \rightarrow s_2 q_2 \rightarrow \ldots \rightarrow s_m \) with \( s_m \in F \). So,

\( q_0 \Rightarrow s_1 q_1 \Rightarrow s_1 s_2 q_2 \Rightarrow \ldots \Rightarrow s_1 s_2 \ldots s_m q_m \Rightarrow s_1 s_2 \ldots s_m = s \)

\( s \in L(G) \). So, \( L(M) \subseteq L(G) \).

Conversely, suppose \( s = s_1 s_2 \ldots s_m \in L(G) \).

There there exists a derivation: \( q_0 \Rightarrow s_1 q_1 \Rightarrow s_1 s_2 q_2 \Rightarrow \ldots \Rightarrow s_1 s_2 \ldots s_m q_m \Rightarrow s_1 s_2 \ldots s_m, \) which implies \((q_0, s_1), q_1) \in \delta, ((q_1, s_2), q_2) \in \delta, \ldots, ((q_{m-1}, s_m), q_m) \in \delta \) and \( q_m \rightarrow \epsilon \in R \), implying \( q_m \in F \).

So, \( q_0 \rightarrow s_1 q_1 \rightarrow \ldots \rightarrow s_m q_m \) is an accept path in \( M \). Therefore \( s \in L(M), \) proving \( L(G) \subseteq L(M) \).

We conclude \( L(G) = L(M) = L \) and \( L \) is context-free. \( \blacksquare \)

- An example above showed that \( L = \{ a^n b^m c^{n+m} : m, m \geq 0 \} \) is context-free but not regular.

So, regular languages are a proper subset of context-free languages.

- Definition: A right-regular context-free grammar is a context-free in which all production rules have the form \( A \rightarrow xb \) for \( A, B \in V \) and \( x \in \Sigma^* \) or \( A \rightarrow \epsilon \).

A right-regular context-free grammar generates a regular language... (use additional states in an NFA to transition through \( |x| > 1 \) in \( A \rightarrow xb \))

- Definition: A context-free grammar \( G = (V, \Sigma, S, R) \) is Chomsky-Normal-Form if every rule in \( R \) has one of the following forms:

  - \( A \rightarrow BC, \) where \( A, B, C \in V \) and \( B \neq S, C \neq S \)
  - \( A \rightarrow a, \) where \( A \in V, a \in \Sigma \)
  - \( S \rightarrow \epsilon, \) where \( S \) is the start variable.

- Advantages of CNF:

  - Every node in a parse tree has two children — except nodes that directly deliver a terminal.
  - Every derivation (except \( S \rightarrow \epsilon \), if present) is strictly expanding. That is, each succeeding sentential form is at least as long as its predecessor. There are no derivations that oscillate in size.

- Five-step process to transform a give CFG into CNF. (Lower-case symbols below are general sentential forms.)

  - Remove the start symbol from the right side of rules. How? Create a new start symbol, say \( S_1 \), expand \( V \) to \( V \cup \{ S_1 \} \), expand \( R \) to \( R \cup S_1 \rightarrow S \), where \( S \) is the original start symbol.
  - Remove all \( \epsilon \)-productions, which are productions of the form \( A \rightarrow \epsilon \), where \( A \) is not the start variable. How? Follow the process below subject to the constraints that we never restore a rule that has been previously removed and we never duplicate an existing rule.
while (\(\exists\epsilon\)-production of the form \(A \rightarrow \epsilon\))\{
  remove rule \(A \rightarrow \epsilon\) from \(R\)
  for each rule with RHS containing \(k \geq 1\) symbols \(A\) (say \(u_1Au_2A\ldots u_kAu_{k+1}\))
    for \((j = 1\) to \(k)\)
      add \(\binom{k}{j}\) rules to \(R\), obtained by dropping \(j\) A’s from the RHS
}\)

- Remove all unit-productions, which are productions of the form \(A \rightarrow B\). How? Follow the process below subject to the constraints that we never restore a rule that has been previously removed and we never duplicate an existing rule.
  while (\(\exists\) a unit-production of the form \(A \rightarrow B\)\{
    remove rule \(A \rightarrow B\) from \(R\)
    for each rule \(B \rightarrow u\)
      add \(A \rightarrow u\) to \(R\)
  }\)

- Remove all rules having more than two symbols on the RHS. How? Introduce new variables as in this example.
  Suppose \(A \rightarrow u_1u_2\ldots , u_k\). Replace with
  \[
  \begin{align*}
  A & \rightarrow u_1A_1 \\
  A_1 & \rightarrow u_2A_2 \\
  A_2 & \rightarrow u_3A_3 \\
  \vdots & \vdots \\
  A_{k-3} & \rightarrow u_{k-2}A_{k-2} \\
  A_{k-2} & \rightarrow u_{k-1}u_k
  \end{align*}
  \]
  in which \(A_1, \ldots, A_{k-2}\) are new elements of \(V\).

- Remove all rules of the form \(A \rightarrow u_1u_2\), where \(u_1, u_2\) are not both in \(V\). How?
  Disguise all terminals as new variables. For example, for \(a \in \Sigma\), introduce \(V_a \in V\) and \((V_a \rightarrow a) \in R\).
  Now, if a rule is \(A \rightarrow ab\) or \(Xb\) or \(aX\), replace with \(A \rightarrow V_aV_b\) or \(XV_b\) or \(V_aX\).

- Theorem: Let \(\Sigma\) be an alphabet and let \(L \subseteq \Sigma^*\) be a context-free language. Then there exists a context-free grammar \(G\), in Chomsky-Normal-Form, such that \(L = L(G)\).
  Proof: CNF construction above. □

- Example: \(G = (V = \{A, B\}, \Sigma = \{0, 1\}, A, R)\), where \(R = \{\begin{align*}
  A & \rightarrow BAB|B|\epsilon \\
  B & \rightarrow 00|\epsilon
  \end{align*}\) .
  - Eliminate start symbol from right sides:
    \[
    \begin{align*}
    S & \rightarrow A \\
    A & \rightarrow BAB|B|\epsilon \\
    B & \rightarrow 00|\epsilon
    \end{align*}
    \]
Eliminate $\epsilon$-rules:

Drop $A \rightarrow \epsilon$, Add \[
\begin{align*}
S & \rightarrow A \epsilon \\
A & \rightarrow BB \\
B & \rightarrow 00 \epsilon
\end{align*}
\]
giving \[
\begin{align*}
S & \rightarrow A | \epsilon \\
A & \rightarrow BAB | BB | AB | BA \\
B & \rightarrow 00
\end{align*}
\]

Drop $B \rightarrow \epsilon$, from $A \rightarrow BAB$, add \[
\begin{align*}
S & \rightarrow A \\
A & \rightarrow BAB | B | BB | AB | BA \\
B & \rightarrow 00
\end{align*}
\]
Drawable terminals: introduce $T \rightarrow 0$, force right-hand-sides to two symbols with $X = AB$ giving \[
\begin{align*}
T & \rightarrow 0 \\
X & \rightarrow AB \\
S & \rightarrow \epsilon | BX | BB | AB | BA | TT \\
A & \rightarrow BX | BB | AB | BA | TT \\
B & \rightarrow TT
\end{align*}
\]

- Eliminate unit productions:

Drop $A \rightarrow A$ (no consequences) giving \[
\begin{align*}
S & \rightarrow A | \epsilon \\
A & \rightarrow BAB | BB | AB | BA \\
B & \rightarrow 00
\end{align*}
\]

Drop $S \rightarrow A$, add $S \rightarrow BAB | BB | AB | BA$, giving \[
\begin{align*}
S & \rightarrow \epsilon | BAB | BB | AB | BA \\
A & \rightarrow BAB | BB | AB | BA \\
B & \rightarrow 00
\end{align*}
\]

Drop $S \rightarrow B$, add $S \rightarrow 00$, giving \[
\begin{align*}
S & \rightarrow \epsilon | BAB | BB | AB | BA | 00 \\
A & \rightarrow BAB | BB | AB | BA \\
B & \rightarrow 00
\end{align*}
\]

Drop $A \rightarrow B$, add $A \rightarrow 00$, giving \[
\begin{align*}
S & \rightarrow \epsilon | BAB | BB | AB | BA | 00 \\
A & \rightarrow BAB | BB | AB | BA | 00 \\
B & \rightarrow 00
\end{align*}
\]

- Lemma. Let $G$ be a context-free grammar in Chomsky Normal Form. Let $s \neq \epsilon$, $s \in L(G)$. Let $T$ be a parse tree for $s$. Let $h$ be the height of $T$ (the number of links in the longest path from the root). Then $|s| \leq 2^{h-1}$.

Proof: Starting with the root at level 0, the maximum possible nodes at any level $d$ is $2^d$. If the tree has depth (height) $h$, then the variable that delivers its single terminal (a pre-leaf) occurs at a level $h - 1$ or less. The longest string in the leaves is then $2^{h-1}$ or less (if some final variables appear at lower levels). That is, $|s| \leq 2^{h-1}$.

- Theorem (Pumping Lemma for Context-free Languages). Let $L$ be a context-free language. Then there exists an integer $p \geq 1$ (the pumping length) such that every string $s \in L$ with $|s| \geq p$ can be written as $s = uvxyz$ where

1. $|vy| \geq 1$. That is, $v$ and $y$ are not both empty
2. $|vxy| \leq p$, and
3. $uv^ixy^iz \in L$, for all $i \geq 0$.

Proof: Let $G = (V, \Sigma, S, R)$ be a Chomsky Normal Form context-free grammar for $L$. Let $p = 2^r$, where $r = |V|$.
Suppose \( s \in L(G) \) with \( |s| \geq p \). Let \( T \) be a parse tree for \( s \) of height (depth) \( h \). Then we have

\[
2^r = p \leq |s| \leq 2^{h-1} \\
r \leq h - 1 \\
h \geq r + 1.
\]

The longest path from root to leaf contains \( h \) edges and therefore contains \( h + 1 \) nodes, all of which are variables, except the last, which is a terminal.

Hence there are \( h \) variables on this path. And \( h \geq r + 1 \). As there are only \( r \) variables in the grammar, the pigeonhole principle forces a repetition along this path.

Consider the last \( r + 1 \) variables on the path, and the bottom-most repetition within these variables.

![Diagram of a parse tree](image)

Tree with yield \( vxy \) has height \( h_0 \leq h - \lfloor h - (r + 1) \rfloor = r + 1 \), which implies (above lemma) \( |vxy| \leq 2^{r+1-1} = 2^r = p \).

Also, lower repetition may be on a leftmost path from the upper repetition, or it might be on a rightmost path, or somewhere between the two. Any those cases, we have \( |v| = 0, |y| > 0, \) or \( |v| > 0, |y| = 0, \) or \( |v| \geq 0, |y| > 0, \) respectively. So, in all cases, \( |vy| \geq 1 \).

Can remove tree anchored at \( U \) and replace it with tree anchored at \( L \), giving yield \( uxz \).

Can copy tree anchored at \( U \), remove tree anchored at \( L \), and re-anchor U-tree at \( L \), giving yield \( uv^2xy^2z \), and so forth.

- Pumping lemma example: The language \( L = \{a^nb^nc^n : n \geq 0\} \), over \( \Sigma = \{a, b, c\} \) is not context-free.

Proof: (contradiction). Suppose \( L \) is context-free.

Then the pumping lemma applies. Let \( p \) be the pumping length and consider \( s = a^nb^nc^n \in L \).

Also, \( s = uvxyz \) with \( |uvy| \leq p \), which means segment \( uvy \) cannot contain all three letters. Nevertheless \( vy \) includes at least one letter.

Thus the \( i = 0 \) option, dropping \( uy \) reduces some letters but not others, producing a string not in \( L \).

Similarly, \( i > 0 \) pumping increases the number of at most two of the three letters, producing a string that is not in \( L \). But, the pumping lemma insists that such strings are in \( L \) — a contradiction.

We conclude that \( L \) is not context-free.

- Pumping lemma example: The language \( L = \{ww : w \in \{a, b\}^*\} \) is not context-free.

Proof: (contradiction). Suppose \( L \) is context-free.
Let \( p \) be the pumping length. We need \( s \in L, |s| \geq p \) to pump for a contradiction.

**Wrong choice:** Let \( s = a^p ba^p b \in L \). Then \( s = uvxyz \).

But, suppose \( u = a^{p-1}, v = a, x = b, y = a, z = a^{p-1} b \), in which case pumping does not destroy the symmetry necessary to remain in \( L \).

**Correct choice:** \( s = a^p b^p a^p b^p \in L \). Then \( s = uvxyz \) with \(|uxy| \leq p\), which means that \( uxy \) overlaps at most two adjacent segments of the four \( p \)-segments.

If \( uxy \) lives in a single \( p \)-block, then pumping increases one of the \( a \) or \( b \) spans but not the other, which breaks the symmetry and produces a string that is not in \( L \) — a contradiction.

If \( uxy \) spans two adjacent \( p \)-blocks, these two blocks are an \( a \)-block and a \( b \)-block. Thus pumping, if it manages not to garble the pattern, affects only one of the two \( a \) segments and/or one of the two \( b \) segments, which breaks the symmetry and produces a string that is not in \( L \) — a final contradiction.

We conclude that \( L \) is not context-free.

**Pumping lemma example:** Over \( \Sigma = \{a, b, c\} \), the language \( A = \{a^m b^n c^{mn} : m, n \geq 0\} \) is not context-free.  
Proof: (contradiction). Suppose \( A \) is context-free. Let \( p \) be the pumping length.

Consider \( s = a^p b^p c^p \in A \). Note \(|s| \geq p\), which implies the pumping lemma applies.

\( s = uvxyz \) with \(|uxy| \leq p\) and \( uv^i xy^i z \in A \) for all \( i \geq 0 \).

\( uxy \) can contain at most two of the three alphabet symbols. If \( uxy \) contains only \( a \)'s or only \( b \)'s or only \( c \)'s, then pumping changes only a single segment, thereby creating strings outside \( A \) — a contradiction.

If \( uxy \) contains only \( a \)'s and \( b \)'s, then pumping does not change the \( c \)-count, again creating strings outside \( A \) — a contradiction.

The final possibility is that \( uxy \) contains only \( b \)'s and \( c \)'s.

Then, \( u \) is either empty or contains only \( b \)'s — otherwise pumping would garble \( b \)'s and \( c \)'s, creating strings outside of \( A \).

Likewise \( y \) is either empty or contains only \( c \)'s.

The pumping lemma asserts \(|uy| > 1\), which means \( u \) and \( y \) are not both empty.

If \(|u| = 0\), then pumping adds only \( c \)'s, which destroys the pattern that the \( c \)-count is the product of the \( a \) and \( b \) counts — a contradiction.

If \(|y| = 0\), then pumping adds only \( b \)'s, which likewise destroys the pattern that the \( c \)-count is the product of the \( a \) and \( b \) counts — a contradiction.

So, \( u = b^j, y = c^k \) for \( j, k > 0 \).

Consider dropping \( v \) and \( y \), leaving only \( uxz = a^p b^{p-j} c^{p^2-k} \in A \).

That is, \( p \cdot (p - j) = (p^2 - k) \), giving \( p^2 - pj = p^2 - k \) or \( pj = k \).

As \( j > 1, k = jp > p \), which forces \(|uxy| \geq |vy| = j + k \geq 1 + p\), which contradicts \(|uxy| \leq p\) — a final contradiction.
Definition: A **deterministic push-down automaton** (DPA) is a 5-tuple $M = (Q, \Sigma, \Gamma, q, \delta)$, where

- $Q$ is a **finite** set of **states**
- $\Sigma$ is a **finite** (tape, input) set **symbols**
- $\Gamma$ is a **finite** (stack) set of **symbols**
- $q \in Q$ is the **start** state
- $\delta : Q \times (\Sigma \cup \square) \times \Gamma \to Q \times \{N, R\} \times \Gamma^*$ is a **transition function**.

$\delta(r, a, A) = (r', \sigma, w)$ means “In state $r$ with the tape head over symbol $a$ and $A$ at the top-of-stack, the automaton transitions to state $r'$, moves the tape reading head according to $\sigma$ ($\sigma = N$ implies no move, $\sigma = R$ implies one step to the right), and pushes $w$ onto the stack.

**Note:** $w \in \Gamma^*$ is pushed from left to right, such that the rightmost symbol becomes the new top-of-stack.

Operation of PDA $(Q, \Sigma, \Gamma, q, \delta)$.

- Starting configuration. Machine in state $q$, tape head on the first symbol of the input string, stack contains only $\$$ symbol (end-of-stack).
- Computation and termination. Machine performs the computations directed by $\delta$ and terminates when the stack becomes empty. If the stack does not empty, the machine does not terminate.
- Acceptance. The machine accepts string $a_1a_2 \ldots a_n \in \Sigma^*$ if the machine terminates on this input and at the time of termination the tape resides on the $\square$ at the end of the input string. In all other cases, the machine rejects the string.

Thus the machine rejects if it does not terminate, or if the input head is not over the $\square$ immediately to the right of the input string on termination.

- For DPA $L(M) = \{w \in \Sigma^* : M$ accepts $w\}$.

- For a nondeterministic pushdown automaton (NPA), we change only the transition function to $\delta : Q \times (\Sigma \cup \square) \times \Gamma \to \mathcal{P}_f(Q \times \{N, R\} \times \Gamma^*)$, where $\mathcal{P}_f(K)$ is the collection of all **finite** subsets of $K$.

- An NPA $M$ accepts string $w$ if there exists any accepting trajectory. That is, a choice of transitions such the stack empties, leaving the tape head over the $\square$ just to the right of the input.
• DPA example accepting \( L = \{0^n1^n : n \geq 0\} \subseteq \{0,1\}^* \).
\[ M = (Q, \Sigma, \Gamma, q_0, \delta), \]
where \( \Sigma = \{0,1\}, \Gamma = \{\$, S\}, Q = \{q_0, q_1\} \) and \( \delta \) is as follows.

\[
\begin{align*}
q_00\$ &\rightarrow q_0RSS \quad \text{push } S \text{ onto the stack} \\
q_00S &\rightarrow q_0RSS \quad \text{push } S \text{ onto the stack} \\
q_01\$ &\rightarrow q_0NS \quad \text{first symbol is 1; loop forever} \\
q_01S &\rightarrow q_1R\epsilon \quad \text{first 1 encountered} \\
q_0\square\$ &\rightarrow q_0N\epsilon \quad \text{empty input string accepted} \\
q_0\square S &\rightarrow q_0NS \quad \text{input contains only zeros; loop forever} \\
q_10\$ &\rightarrow q_1NS \quad \text{discover 0 right of a 1; loop forever} \\
q_10S &\rightarrow q_1NS \quad 0 \text{ right of 1; loop forever} \\
q_11\$ &\rightarrow q_1NS \quad \text{too many 1s; loop forever} \\
q_11S &\rightarrow q_1R\epsilon \quad \text{pop S from the stack} \\
q_1\square\$ &\rightarrow q_1N\epsilon \quad \text{accept} \\
q_1\square S &\rightarrow q_1NS \quad \text{too many 0s; loop forever}
\end{align*}
\]

• Example NPA accepting \( L = \{vwv : v, w \in \{a,b\}^*, |v| = |w|\} \subseteq \{a,b\}^* \).

Let \( M = (Q = \{q, q'\}, \Sigma = \{a,b\}, \Gamma = \{\$, S\}, q, \delta) \), where \( \delta \) is as follows.

\[
\begin{align*}
qa\$ &\rightarrow qaRSS \quad \text{push } S \text{ onto the stack} \\
qaS &\rightarrow qRSS \quad \text{push } S \text{ onto the stack} \\
qb\$ &\rightarrow q'R\$ \quad \text{might be the middle, with empty string to its left} \\
qb\$ &\rightarrow qRSS \quad \text{might not be the middle} \\
qbS &\rightarrow q'RS \quad \text{might be the middle, don’t count this b in matching string to the left} \\
qbS &\rightarrow qRSS \quad \text{might not be the middle} \\
q\square\$ &\rightarrow qN\$ \quad \text{reached end of string with discovering the middle; loop forever} \\
q\square S &\rightarrow qNS \quad \text{ditto} \\
q'a\$ &\rightarrow q'N\epsilon \quad \text{no match from left part; terminate but still reject (not over the } \square) \\
q'aS &\rightarrow q'R\epsilon \quad \text{match input with an } S \text{ from the stack} \\
q'b\$ &\rightarrow q'N\epsilon \quad \text{no match from left part; terminate but still reject} \\
q'bS &\rightarrow q'R\epsilon \quad \text{match input with an } S \text{ from the stack} \\
q'\square\$ &\rightarrow q'N\epsilon \quad \text{accept} \\
q'\square S &\rightarrow q'NS \quad \text{loop forever}
\end{align*}
\]

Remark: It can be shown that there is no DPA that accepts \( L \), which implies that NPA’s are more powerful than DPA’s, unlike the case with DFA and NFA.

• Theorem: Let \( \Sigma \) be an alphabet and let \( L \in \Sigma^* \) be a language. Then \( L \) is context-free if and only if there exists an NPA that accepts \( L \).

Proof: (\( \Rightarrow \))

Let \( L \) be context-free. Then there exists a Chomsky-Normal-Form context-free grammar \( G = (V, \Sigma, \$ \in V, R) \) with \( L = L(G) \).

Every rule in the grammar has one of three forms: \( A \rightarrow BC \) (all variables), \( A \rightarrow a \), a single terminal, \( \$ \rightarrow \epsilon \).

We define a NPA \( M = (Q = \{q\}, \Sigma, \Gamma = V, q, \delta) \), where \( \delta \) is contructed from \( G \) as follows.

\[
\begin{align*}
- A \rightarrow BC &\in R \text{ forces } \delta(q,a,A) = (q, N, CB) \text{ for all } a \in \Sigma \\
- A \rightarrow a &\in R \text{ forces } \delta(q,a,A) = (q, R, \epsilon) \\
- \text{If } \$ &\rightarrow \epsilon \in R, \text{ then } \delta(q, \square, \$) = (q, N, \epsilon).
\end{align*}
\]

Verify \( \$ \Rightarrow a_1a_2 \ldots a_{i-1}A_kA_{k-1} \ldots A_1 \) iff \( M \) proceeds from
which implies \( w \in L(G) \) iff \( w \in L(M) \).

We can do this verification for an “almost-Chomsky” grammar, where all productions have the form \( X \rightarrow Y_1Y_2\ldots Y_n \) for \( n \geq 2 \) or \( X \rightarrow a \in \Sigma \).

Such a rule places \( \delta(q,a,X) = (q,N,Y_n\ldots Y_1) \) in the machine transition function.

For example: \( L = \{a^n b^n : n \geq 0\} \subseteq \{a,b\}^* \).

\( G = (V = \{\$, A, B\}, \Sigma = \{a, b\}, \$, R = \{\$ \rightarrow \epsilon | A\$B, A \rightarrow a, B \rightarrow b\}). \)

On \( w = aabb \), accepting derivation is \( \$ \rightarrow A\$B \rightarrow a\$B \rightarrow aA\$BB \rightarrow aa\$BB \rightarrow aaBB \rightarrow aaBb \rightarrow aabb \).

Accepting machine trajectory is (stack top to the left, current tape position at underlined position)

<table>
<thead>
<tr>
<th>Tape</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>( aabb ) ( \square ) $</td>
<td>initial position</td>
</tr>
<tr>
<td>( aabb ) ( \square ) ( A$B )</td>
<td>via ( $ \rightarrow A$B )</td>
</tr>
<tr>
<td>( aabb ) ( \square ) ( $B )</td>
<td>via ( A \rightarrow a )</td>
</tr>
<tr>
<td>( aabb ) ( \square ) ( A$BB )</td>
<td>via ( $ \rightarrow A$B )</td>
</tr>
<tr>
<td>( aabb ) ( \square ) ( $BB )</td>
<td>via ( A \rightarrow a )</td>
</tr>
<tr>
<td>( aabb ) ( \square ) ( BB )</td>
<td>via ( $ \rightarrow \epsilon )</td>
</tr>
<tr>
<td>( aabb ) ( \square ) ( B )</td>
<td>via ( B \rightarrow b )</td>
</tr>
<tr>
<td>( aabb ) ( \square ) ( \epsilon )</td>
<td>via ( B \rightarrow b )</td>
</tr>
</tbody>
</table>

Conversely, (\( \Leftarrow \)) — proof deferred to CS 401...  ■