Lecture Notes

CS259 - Formal Languages

Intro

Sipser[1], Hopcroft[2] `[1.5]`

1. **Alphabet** Σ is finite non-empty set of **symbols/letters**, so the empty string isn't in it $\epsilon \notin \Sigma$.

2. **String/word** w is finite sequence of symbols chosen from some alphabet. **Empty** string ϵ doesn't contain symbols.

3. **Define** Σ^k to be set of strings of length k. E.g. $\Sigma = \{0, 1\}$: $\Sigma^0 = \{\epsilon\}$, $\Sigma^1 = \{0, 1\}$, $\Sigma^2 = \{00, 01, 10, 11\}$ etc.

   Define $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup ...$ to be a set of all strings over an alphabet Σ. Set of all **non-empty** strings $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$

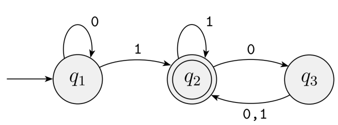
4. **Length** of string w is denoted by $|w|$, importantly $|\epsilon| = 0$. **Substring** is a *consecutive* subsequence within a string. **Concatenation** of strings x, y is denoted by xy.

5. **Language** is some $L \subseteq \Sigma^*$. Decision problem is a function $w \in \Sigma^* \to \{\text{Yes}, \text{No}\}$. Not all languages have algorithms.

6. **Finite Automata/Machine (FA)** $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ is quintuple with finite set of states Q, alphabet Σ, state transition function δ, initial state q_0 and set of all accepting states $F \subseteq Q$.

Deterministic Finite Automata DFA

1. **Deterministic Finite Automata/Machine (DFA)** is FA with single-choice transition function $\delta : Q \times \Sigma \to Q$



2. **Extended transition function** $\hat{\delta} : Q \times \Sigma^* \to Q$ says what happens if you start in any state q and follow any sequence of inputs given transition function δ. Define by induction on length of input str $w{=}xa$, $|\epsilon|{=}0$, $|xa| = |x|{+}|a|$:

   Base Case: $\hat{\delta}(q, \epsilon) = q$. No inputs read, no state change.
   I.S.: $\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a)$ with $w = xa$, tail recursion.

3. **Language** $L(\mathcal{A}) = \{w \in \Sigma^* : \hat{\delta}(q_0, w) \in F\}$ of automaton $\mathcal{A}$ is set of all strings w accepted by $\mathcal{A}$, or $q_{final} \in F$.
   Regular language (RL) is one recognised by some FA.

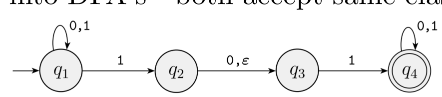
4. Collection of objects in domain D is **closed** under an operation $\Box$ if $x_1, x_2 \in D \Rightarrow x_1 \Box x_2 \in D$.

5. Let A, B be languages. Class of RL is closed under following for $A = (Q_A, \Sigma, q_A, F_A, \delta_A)$, $B = (Q_B, \Sigma, q_B, F_B, \delta_B)$:

   - **Intersection** $A \cap B = \{x : x \in A \land x \in B\}$
     $M' = (Q_A \times Q_B, \Sigma, (q_A, q_B), F_A \times F_B, \delta')$ where δ' is:
     $\forall a \in \Sigma,\ x \in Q_A,\ y \in Q_B :\ \delta'((x, y), a) = (\delta_A(x, a), \delta_B(y, a))$
   - **Union** $A \cup B = \{x : x \in A \lor x \in B\}$
     $M' = (Q_A \times Q_B, \Sigma, (q_A, q_B), (F_A \times Q_B) \cup (Q_A \times F_B), \delta')$
     $\forall a \in \Sigma,\ x \in Q_A,\ y \in Q_B :\ \delta'((x, y), a) = (\delta_A(x, a), \delta_B(y, a))$
   - **Complementation** $\overline{A} = \Sigma^* \setminus A$
     $M' = (Q_A, \Sigma, q_A, Q_A \setminus F_A, \delta_A)$
   - **Concatenation** $A \circ B = \{xy : x \in A \land y \in B\}$
     $M' = (Q_A \cup Q_B, \Sigma \cup \{\epsilon\}, q_A, F_B, \delta')$ where δ' is:
     $\forall a \in \Sigma,\ x \in Q_A :\ \delta'(x, a) = \delta_A(x, a)$,
     $\forall a \in \Sigma,\ y \in Q_B :\ \delta'(y, a) = \delta_B(y, a)$,
     $\forall f \in F_A :\ \delta'(f, \epsilon) = \{q_B\}$
   - **Set Difference** $A \setminus B = A \cap (\Sigma^* \setminus B)$
     $M' {=} (Q_A \times Q_B, \Sigma, (q_A, q_B), F_A \times (Q_B \setminus F_B), \delta')$ where δ' is:
     $\forall a \in \Sigma,\ x \in Q_A,\ y \in Q_B :\ \delta'((x, y), a) = (\delta_A(x, a), \delta_B(y, a))$
   - **Kleene Star** $A^* = \{x_1 x_2 \cdots x_k : x_i \in A,\ k \geq 0\}$
     $M' = (Q_A \cup \{q_s\}, \Sigma \cup \{\epsilon\}, q_s, F_A \cup \{q_s\}, \delta')$ where δ' is:
     $\forall a \in \Sigma,\ q \in Q_A :\ \delta'(q, a) = \delta_A(q, a)$,
     $\delta'(q_s, \epsilon) = \{q_A\}$,
     $\forall f \in F_A :\ \delta'(f, \epsilon) = \{q_A, q_s\}$

Nondeterministic Finite Automata NFA

1. NFA's are more succinct than and can always be converted/compiled into DFA's - both accept same class of RL's.



2. $\forall w \in \Sigma$ a DFA has exactly 1 transition out of a state whereas NFA can have 0, 1 or multiple, hence the transition function $\delta : Q \times \Sigma_\epsilon \to 2^Q$ where $\Sigma_\epsilon = (\Sigma \cup \{\epsilon\})$.

3. **Extended transition function** $\hat{\delta} : Q \times \Sigma^* \to 2^Q$ also defined by induction on length of input string $w = xa$:

   Base Case: $\hat{\delta}(q, \epsilon) = q$. No inputs read, no state change.
   I.S.: $\hat{\delta}(q, w) = \cup_{i=1}^{k} \delta(p_i, a)$, where $\hat{\delta}(q, x) = \{p_1, ..p_k\}$

   Informally, find $\hat{\delta}(q, w)$ by first computing left part $\delta(q, x)$, and for each resulting state p_i, finding $\hat{\delta}(p_i, a)$ where a is last symbol of w.

[1] Introduction to the Theory of Computation, 3rd ed.
[2] Hopcroft Motwani Ullman 2014

4. **Language** of NFA $L(\mathcal{A}) = \{w : \hat{\delta}(q_0, w) \cap F \neq \varnothing\}$
NFA $\mathcal{A}{=}(Q, \Sigma, \delta, q_0, F)$ accepts str $w = w_1 w_2 \cdots w_m \in \Sigma$ if it's possible to make any sequence of choices of next state $q_0, q_1, ..q_m \in Q$ while reading chars $w_i \in w$, where $q_{i+1} \in \delta(q_i, w_{i+1})$ and go from start state q_0 to any accepting state $q_m \in F$. $\exists$ **accepting run** on word w.

5. **Theorem**: Every NFA has an equivalent DFA.
+: Language regular iff recognised by some NFA.
+: DFA $\mathcal{D}$ constructed from NFA $\mathcal{N} \Rightarrow L(\mathcal{D}) = L(\mathcal{N})$
NFA $\mathcal{N} = (Q, \Sigma, \delta, q_0, F) \rightarrow$ DFA $\mathcal{D} = (2^Q, \Sigma, \delta', \{q_0\}, F')$
where $\delta'(R, a) = \cup_{r \in R} \delta(r, a)$ for set $R \subseteq Q$ of original states, $F' = \{R \subseteq 2^Q : R \cap F \neq \varnothing\}$ contains accept state.

Epsilon-Closure (EClose/ϵ-Close)

1. **Epsilon-transition** $\delta(q, \epsilon)$ denotes empty-string transition yielding unconditionally reachable states. Useful for proving equivalence of RL classes.

2. **ϵ-NFA** $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ but with $\delta(q,\ w_i \in \Sigma \cup \{\epsilon\})$ which can accept any character including an empty string.

3. Epsilon-Closure **EClose**(q): $Q \rightarrow 2^Q$ recursively defines all states reachable from q with ϵ-transitions alone:
Base Case: state $q \in \text{EClose}(q)$ (stays itself)
I.S.: if state $p \in \text{EClose}(q)$ and $\exists$ transition $\delta(p, \epsilon) = R$ of all reachable states r_i then $\forall r_i \in R : r_i \in \text{EClose}(q)$.

4. **ϵ-NFA extended transition func.** $\hat{\delta}(q, w) : Q \times \Sigma^* \rightarrow 2^Q$ produces all states $R \subseteq Q$ to which $\exists$ a **run** from state q upon reading **string** $w{=}xa$ with nonempty last char $a \in \Sigma \neq \epsilon$.
Base Base: $\forall q \in Q : \hat{\delta}(q, \epsilon) = \text{EClose}(q)$ (by definition).
I.H.: let $\hat{\delta}(q, x) = P$ of states reachable from q by following sequence x and $\cup_{i=1}^{k} \hat{\delta}(p_i \in P, a){=}R$ of states reachable from previous step following final non-empty input a. Finally, define $\hat{\delta}(q, w) = \cup_{j=1}^{m} \text{EClose}(r_j \in R)$.

5. **Theorem**: Every ϵ-NFA has an equivalent DFA. $Q' \subseteq Q$
ϵ-NFA $\mathcal{E} = (Q, \Sigma, \delta, q_0, F) \rightarrow$ DFA $\mathcal{D} = (Q', \Sigma, \delta\, q_0, F')$.

6. **Theorem**: For every ϵ-free NFA $N = (Q, \sigma, q_0, F, \delta)$: $\exists$ DFA $D = (2^Q, \Sigma, \text{EClose}(q_0), F_D, \delta_D)$ s.t. $L(N) = L(D)$.
Proof: Given $S_D : 2^Q \times \Sigma \rightarrow 2^Q$ take $a \in \Sigma$, $S \in 2^Q$ ($S \subseteq Q$), suppose $S = \{s_1, .., s_m\}$,then $\delta_D(S, a) = \cup_{i=1}^{m} \delta(s_i, a)$. Now, $F_D = \{A \subseteq Q : A \cap F \neq \varnothing\}$.

Regular Expressions (RegEx)

1. Regular expression $R \in \{a \in \Sigma, \epsilon, \varnothing, R_1{+}R_2, R_1{\circ}R_2, R_1^*\}$. Order of operations: 1. Kleene $*$ 2. Concat $\circ$ 3. Union $+$. E.g. "all languages with second to last character being 1" is $(0+1)^* \circ 1 \circ (0+1)$.

2. **Remember**: ϵ represents a language containing only the empty string, $\varnothing$ represents the language that doesn't contain any strings. Empty word ϵ is **something**, empty state $\varnothing$ is **nothing**, hence $L(R \circ \epsilon) = L(R)$, but $L(R \circ \varnothing) = \varnothing$.

3. **Remember**: Languages L contain strings, alphabets Σ contain symbols, so $L^1 \neq \Sigma^1$ but $(L^1)^*$ and $(\Sigma^1)^*$ denote the same language. Remember: $L(\varnothing^*) = \{\epsilon\}$

4. **Kleene star/plus** $()^{*/+}$ creates any number k of concatenated ordered values $A^*{=}\{x_1, x_2, ..x_k : k \geq 0/1, \forall x_1 \in A\}$ or infinite union $A^* = A^0 \cup A^1 .. \cup A^k$. But finite $\varnothing^*{=}\{\epsilon\}$.

5. **Token** is elemental object of programming language (keywords: if, else ..; vars; integers etc.) E.g. Integers : $("+" + "-" + \epsilon) \circ (1 + 2 + .. + 9) \circ (0 + 1 + .. + 9)^*$.

6. **Theorem**: Language is regular iff some regular expression describes (generates) it. NFA $\equiv$ DFA $\equiv$ RegEx.

7. **Regex $\rightarrow$ NFA**: node $\circ$, accepting $\bullet$
   - $L(R) = \{a\}$ if $R = a \in \Sigma$ $\quad \rightarrow \circ \xrightarrow{a} \bullet$
   - $L(R) = \{\epsilon\}$ if $R = \epsilon$ $\quad \rightarrow \bullet$
   - $L(R) = \varnothing$ if $R = \varnothing$ $\quad \rightarrow \circ$
   - $L(R) = L(R_1) \cup L(R_2)$ if $R = R_1 + R_2$ $\quad \rightarrow \circ \xrightarrow{\epsilon} N_1, N_2$
   - $L(R) = L(R_1) \circ L(R_2)$ if $R = R_1 \circ R_2$ $\quad N_1 \xrightarrow{\epsilon} N_2$
   - $L(R) = (L(R_1))^*$ if R_1^* $\quad \bullet \xrightarrow{\epsilon} N_1 \xleftarrow{\epsilon} \bullet_1, \bullet_2 ..$

8. Convert **NFA/DFA $\rightarrow$ RegEx**: (*see GNFA*)
   1. add $q_{\text{start}}, q_{\text{final}}$ without changing automaton
   2. eliminate non-final and non-start states
   3. eliminate q_{final}

Generalised NFA (GNFA)

1. **GNFA** $(Q, \Sigma, \delta, q_{\text{start}}, q_{\text{final}})$ is an NFA where each transition may have any RegEx $r_i \in \mathcal{R}$ as label instead of just Σ_ϵ members. Transition $\delta : (Q \backslash \{q_{\text{final}}\}) \times (Q \backslash \{q_{\text{start}}\}) \rightarrow \mathcal{R}$ where $\mathcal{R}$ is set of all RegEx over the alphabet.

2. GNFA has unique q_{final} state (**sink**): $\nexists$ transitions to any other state, q_{start} state (**source**): no transitions to start state. Otherwise $\exists$ pairwise transition between **all** states.

3. Convert **DFA $\rightarrow$ GNFA**: $q'_{\text{start}} \rightarrow_\epsilon q_{\text{start}} \rightarrow \ldots$
   1. add new q'_{start} with ϵ-transition to q_{start}.
   2. add new q'_{final} with ϵ-transition reachable from q_{final}.
   3. replace pairwise multi-transitions with single transition
   4. Add $\varnothing$ transit. between pairs requiring but missing one

4. Convert **GNFA $\rightarrow$ RegEx**: Assume GNFA has $k \geq$ states since have unique $q_{\text{start}}, q_{\text{final}}$. Determine regex and replace with current transition:
If $k > 2$: construct equivalent GNFA with $k - 1$ states removing a non-edge state, repeat until $k = 2$ resulting in RegEx R equivalent to original DFA.

5. To remove non-edge state, remove path $p_1 \to q \to p_2$ s.t. $\forall(q_\alpha, q_\beta) \in (Q \backslash \{q_{\text{final}}, q_1\}) \times (Q \backslash \{q_{\text{start}}, q_1\})$, and have new: $\delta'(q_\alpha, q_\beta) = \delta(q_\alpha, q_\beta) + \delta(q_\alpha, q_1) \cdot \delta(q_1, q_1)^* \cdot \delta(q_1, q_\beta)$

6. GNFA accepts string $w \in \Sigma^*$ if $w = w_1 \cdot w_2 \cdots w_k$ where $\forall w_i \in \Sigma^*$ and a sequence of states $q_0, ..q_k$ exists s.t. $q_0 = q_{\text{start}}, q_k = q_{\text{final}}$ and for each $i : w_i \in L(R_i)$ where $R_i = \delta(q_{i-1}, q_i)$.

7. **Claim**: For any GNFA G the RegEx produced by above method is equivalent to G.
   Proof: by induction on k states in G. Base case: $k = 2$, I.H.: assume true for $k-1$ that G on k states accepts w, then $\exists$ sequence $q_{\text{start}}, q_1, ..q_{\text{final}}$ that w uses. If removed state q is not part of it then new $k-1$ automaton accepts the same word. If q appears as $p_1 qq..p_2$ then removing qs and considering new $k-1$ automaton doesn't change word being accepted since $p_1 \to p_2$ now encapsulates q.

Non-Regularity & Myhil-Nerode

1. Strings $x, y \in \Sigma^*$ are **distinguishable** by language $L \in \Sigma^*$ if $\exists w \in \Sigma^*$ s.t. there's only one of xw or yw in L.

2. **Lemma**: **Indistinguishability** $\equiv_L$ is an equivalence relation on Σ^*. Strings x, y indistinguishable by L if $x \equiv_L y$.

3. **Index** of $\equiv_L$ is the number of its equivalence classes.

4. **Myhil–Nerode Theorem**: Let $L \subseteq \Sigma^*, k \in \mathbb{Z}^+$, then $\equiv_L$ has at most k equivalence classes iff $L = L(\mathcal{A})$ for some DFA $\mathcal{A}$ with at most k states.

   Corollary: $L \subseteq \Sigma^*$ is regular iff $\equiv_L$ has **finite index**, otherwise L is a **non-regular language**.

5. All strings $a^n, n \in \mathbb{Z}$ are distinguishable because $a^i b^i \in L$ but $a^j b^i \in L$ iff $i = j$, so it doesn't always hold.

   Proof: given L, DFA $\mathcal{A} = (Q, \Sigma, q_0, F, \delta)$, $Q = \{q_1, ..q_n\}$. For $x, y \in \Sigma^* : x \sim y$ if $\hat{\delta}(q_0, x) = \hat{\delta}(q_0, y)$ (arrive at the same state from both). So, $\sim_{\mathcal{A}}$ is equiv relation on Σ^*, hence $x \sim_{\mathcal{A}} y \Rightarrow x \equiv_L y$.

6. To prove that language L is **non-regular** need to
   (1) provide infinite set of strings and
   (2) prove that they are pairwise distinguishable by L.
   This will show that they must all lie in distinct equiv classes of $\equiv_L$, so $\equiv_L$ must have an infinite number of equivalence classes, so it's non-regular.

7. To prove that language L is **regular**, it suffices to
   (1) describe the equivalence relation $\equiv_L$ on Σ^*, and
   (2) show that there are finitely many equivalence classes.
   Then, by the Myhill-Nerode theorem, L is regular.
   Or just construct DFA/NFA/Regex accepting L.

Non-Regularity & Pumping Lemma

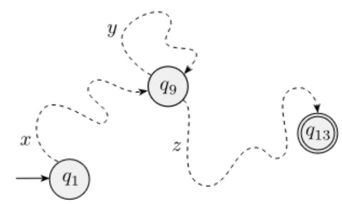
1. **Pigeonhole principle**: if n pigeons are placed into $m \le n$ holes, then some hole has to have more than 1 pigeon in it.

2. For DFA $\mathcal{A}$: $\exists$ a **cycle** reachable from the start state q_{start} that can reach the accept state q_{final} iff $L(\mathcal{A})$ is infinite.

   Equivalently, there must exist a string $w \in L(\mathcal{A})$ of length $|w| > |Q|$ (# states in $\mathcal{A}$) to create that loop.

3. **Pumping Lemma**: Let L be a regular language. Then $\exists$ **pumping length** $p \in \mathbb{Z}^+$ s.t. to account for the loop, any string $w \in L$ with $|w| \ge p$ can be **pumped** (rewritten) as $w = xyz$, where

   1. $\forall i \ge 0 : xy^i z \in L(\mathcal{A})$: middle part y can be repeated any number of times i and remain in the language of $\mathcal{A}$.
   2. $|y| > 0$, $y \ne \epsilon$: middle part cannot be empty, however either x or z may be empty (ϵ).
   3. $|xy| \le p$.



4. FAs have **finite memory**, so non-regular languages use infinite memory and often involve **counting**.

5. **Pumping Lemma proof** that L is non-regular:
   1. Suppose L is regular and let p be L's pumping length.
   2. Choose a string $w \in L$ s.t. $|w| \ge p$.
   3. Let $w = xyz$ be an arbitrary decomposition of w s.t. $|xy| \le p$ and $|y| > 0$.
   4. Find such decomposition, e.g. $x = 0^\alpha, y = 0^\beta, z = 0^\gamma 1^p$ where $\alpha + \beta + \gamma = p$, $\beta > 0$ and $\alpha + \beta \le p$
   5. Pick integer i and argue that $xy^i z \notin L$. $\square$

6. **Myhill-Nerode proof** that L is non-regular:
   1. Suppose, for contradiction, that L is regular.
   2. Consider distinguishing set $S = \{s_n : n \in \mathbb{N}\}$ over Σ^*. I.e. words to be told apart. E.g. $S = \{a^n | n \ge 1\}$.
   3. For any two distinct $s_m, s_n \in S$ with $m \ne n$, find a distinguishing string z such that *exactly one* of $s_m z$ or $s_n z$ is in L. E.g. $z = \{b^n\}$ s.t. $a^n b^n \in L$ but $a^m b^n \notin L$.
   4. Conclude that S is an infinite set of pairwise distinguishable strings, so there are infinitely many equivalence classes of $\equiv_L$.
   5. By the Myhill-Nerode theorem, L cannot be regular.

7. Language L satisfying the Pumping Lemma is not necessarily regular!

Decision Problems

1. **Emptiness**: *In*: DFA A. *Out*: whether $L(A) = \varnothing$.
   How: Perform BFS from the start state q_0 to see if any accepting state is reachable.

2. **Inclusion**: *In*: DFA A_1, A_2. *Out*: if $L(A_1) \subseteq L(A_2)$.
   How: Check if $L(A_1) \cap L(\overline{A_2}) = \varnothing$.

3. **Membership**: *In*: DFA A, alphabet Σ, word $w \in \Sigma^*$. *Out*: whether $(w \in \Sigma^*) \in L(A)$.
   How: Simulate A on w (complexity $O(|E| \cdot |w|)$, where $|E|$ is the number of states/transitions).

Context-free languages (CFL)

1. **Production** or **Rule** $\alpha \to \beta$ is a pair $(\alpha, \beta) \in R$, or line in the grammar that defines transformations for variables. Both $\alpha \neq \epsilon, \beta \in (V \cup \Sigma)^*$ therefore set of productions $R \subseteq (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$ must be finite, $(+)$ because $\alpha \neq \epsilon$

2. **Grammar** $G = (V, \Sigma, R, S)$ with finite mutually-exclusive sets V of variables (nonterminal symbols) and Σ of terminal symbols. R is the finite set of production rules, $S \in V$ is the START variable ("axiom").

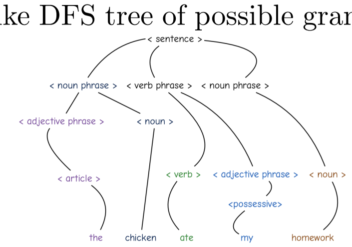
3. **Context-Free Grammar (CFG)**: for every production the start point is within original V, or $\forall(\alpha, \beta) \in R : \alpha \in V$

4. For $x, y \in \Sigma^*$, write "x **yields** y": $\alpha \Rightarrow \beta$ if α can be rewritten as β by applying a production rule $(\alpha, \beta) \in R$. e.g. $G = (\{S\}, \{0, 1\}, R, S)$ rules $R : S \to 0S1$; and $S \to \epsilon$, giving $S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 0011$

5. "x **derives** y": $x \overset{*}{\Rightarrow} y$ iff $\exists$ finite sequence $x_0, x_1, ..x_k, k \geq 0$ s.t. $x_0 = x, x_k = y$ and $\forall i = 0, 1, .., k-1 : x_i \Rightarrow x_{i+1}$, or if y is derivable from x by some sequence of productions. $\overset{*}{\Rightarrow}$ is also a reflexive and transitive closure of $\Rightarrow$.

6. **Language of a grammar** $L(G) = \{w \in \Sigma^* : S \overset{*}{\Rightarrow} w\}$ is the set of all strings in Σ^* which can be derived from S using finitely many applications of production rules in G.

7. **Parse Tree** is like DFS tree of possible grammars' values



Left-most derivation: yields the "left-most" non-terminating variable at each step. E.g. `1A10B` would have to expand (yield) non-terminal A first.

8. **Ambiguous** grammar G iff there are ≥ 2 parse trees for some $w \in L(G) \Leftrightarrow$ there are ≥ 2 leftmost derivations for some $w \in L(G)$. I.e. can generate the same string with multiple parse trees. **Inherently ambiguous** G if every possible CFG that generates this language is ambiguous. I.e. can't rewrite as an equivalent unambiguous grammar.

9. **Chomsky Hierarchy of Grammars**:
   Type 3: Regular **Right linear** $A \to xB$ and **Left linear** $A \to Bx$, and possibly terminal $A \to x$.
   Type 2: Context-free $Q \to w$
   Type 1: Context-sensitive $\alpha A \gamma \to \alpha \beta \gamma$
   Type 0: Recursively-enumerable $a \to \beta$ if α non-empty.
   For variables $A, B \in V$, combinations of V-variables and Σ-characters $\alpha, \beta, \gamma, w \in (V \cup \Sigma)^*$; terminal string $x \in \Sigma^*$.

10. **Strictly Right/Left-linear** grammar has $y \in \Sigma \cup \{\epsilon\}$, NOT Σ^*. Have $T \to yB/By/y$.

11. **DFA $\to$ CFG**: convert DFA $\mathcal{A}$ into equivalent CFG:
    1. Make variable R_i for each state $q_i \in \mathcal{A}$ with start variable R_0 of the grammar representing q_0 start state of $\mathcal{A}$.
    2. Add rule $R_i \to aR_j$ to the CFG if $\delta(q_i, a) = q_j$ is a transition in $\mathcal{A}$ (express transition rules as productions).
    3. Add rule $R_i \to \epsilon$ if q_i is an accept state of the DFA $\mathcal{A}$.

    DFA $\to$ strictly right-linear grammar: for each state q:
    all strings that will take me from q to a final state

    DFA $\to$ strictly left-linear grammar: for each state q:
    all strings that will take me to q from start state

Push-Down Automata (PDA)

1. **Push-Down Automaton (PDA)** $\mathcal{P} = (Q, \Sigma, \Gamma, q_0, F, \delta)$ has a stack Γ that is like a "to-do list" of the automaton. Importantly, it is **nondeterministic** (NFA with a stack).

2. **Configuration** of a PDA $A = (Q, \Sigma, \Gamma, q_0, F, \delta)$ is a pair $(q, s) \in Q \times \Gamma^*$, where q is the current state and s is current stack content (with the top of the stack at the left).

3. Transition function $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to \mathcal{P}(Q \times \Gamma^*)$. The pair $(q', \gamma') \in \delta(q, a, \gamma)$ means a $a, \gamma \to \gamma'$ transition of:
   1. start in state $q \in Q$.
   2. consume next input symbol (condition) $a \in \Sigma_\epsilon$
   3. pop $\gamma \in \Gamma_\epsilon$ from the top of the stack
   4. push a string $\gamma' \in \Gamma^*$ on top of the stack
   5. end up in state $q' \in Q$.

4. A **run** of PDA $A = (Q, \Sigma, \Gamma, q_0, F, \delta)$ on $w \in \Sigma^*$ is a sequence of configurations $(q_0, s_0), ..(q_m, s_m)$ where $(q_i, s_i) \in Q \times \Gamma^*$ for which there exist $w_1, ..w_m \in \Sigma_\epsilon$ s.t.t $w_1, ..w_m = w$ and moreover $s_0 = \epsilon$; and for $i = 1, ..m$, it holds that $s_{i-1} = \gamma s'$, $s_i \gamma' s'$ for $s' \in \Gamma^*$.

5. **Read** ϵ: move without reading next symbol in input.
Push ϵ: don't push anything on the stack.
Pop ϵ: don't pop anything from the stack.

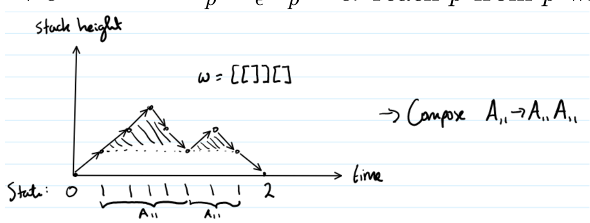
6. **Theorem (CFG $\to$ PDA)**: CFG $G{=}(V,\Sigma,R,S)$ & PDA $P{=}(Q,\Sigma,\Gamma,q_0,F,\delta)$ recognise same lang : $L(P){=}L(G)$. Stack $\Gamma = V \cup \Sigma \cup \{\perp\}$, where $\perp \notin V \cup \Sigma$.

   Push($\perp$), Push(S) then for each rule $A \to \alpha$ in R: pop(A), push (α_k)... push(α_1), upside-down to keep the stack order. Finally, to accept, Pop($\perp$).

7. **Normalised PDA** P:
1. P has a unique accepting state q_{final}
2. P empties its stack before accepting ($\perp$).
3. Whenever $(q',\gamma') \in \delta(q,a,\gamma)$, either $\gamma = \epsilon \vee \gamma' = \epsilon$: either push or pop, but not both.

8. **Theorem (PDA $\to$ CFG)**: For every **normalised** PDA P, there $\exists$ CFG G s.t. $L(P) = L(G)$. Construction proof:
   - Vocabulary $V = \{A_{pq} : p,q \in Q\}$
   - Start variable $s = A_{if}$ with initial i and final f rules.
   - Rules R, $\forall r,p,q \in Q : A_{pq} \to A_{pr}A_{rq}$ in Q^2.

   1. $A_{pq} \to A_{pr}A_{rq}$ $\quad \circ_p \rightsquigarrow \circ_q \equiv \circ_p \rightsquigarrow \circ_r \rightsquigarrow \circ_q$
   2. $A_{pq} \to aA_{p'q'}b$ $\quad \circ_p \rightsquigarrow \circ_q \equiv \circ_p \rightsquigarrow_{+\gamma}\rightsquigarrow \circ_{p'} \rightsquigarrow \circ_{q'} \rightsquigarrow_{-\gamma} \circ_q$
   3. $A_{pq} \to a$ $\quad \circ_p \rightsquigarrow_a \circ_q \equiv a$: reach q from p with a
   4. $A_{pp} \to \epsilon$ $\quad \circ_p \rightsquigarrow_\epsilon \circ_p \equiv \epsilon$: reach p from p with ϵ



context free

1. **Theorem**: for every CFG G there is a CFG G' in CNF s.t. $L(G){=}L(G')$.
Proof (Algorithm): $u,v,w \in V \cup \Sigma$ var $\wedge/\vee$ terminal
1. Add new start variable S_0 and rule $S_0 \to S$ s.t. S_0 doesn't appear on the RHS.
2. Remove each ϵ-rule of form $A \to \epsilon$ where $A \neq S_0$. Now add all possible substitutions of $A = \epsilon$ to R, e.g.
if $R \to uAvAw$, then add 3 rules: $uAvw, uvAw, uvw$
Remember about transitivity: unless removed before,
whenever $A \to B \to \epsilon$ appears, add $A \to \epsilon$
3. Remove all unit rules $A \to B$, then
whenever $B \to u$ appears, add $A \to u$
4. Finally, replace all rules $A \to u_1, u_2 ... u_k : k \geq 3$, where each u_i is var/terminal, with rules
$A \to u_1A_1, \quad A_1 \to u_2A_2 \quad ... \quad A_{k-2} \to u_{k-1}u_k$
5. Replace any terminal u_i in preceding rules with new variable U_i and add rule $U_i \to u_i$.
e.g. $R \to aB \Rightarrow R \to U_1B, \quad U_1 \to a$

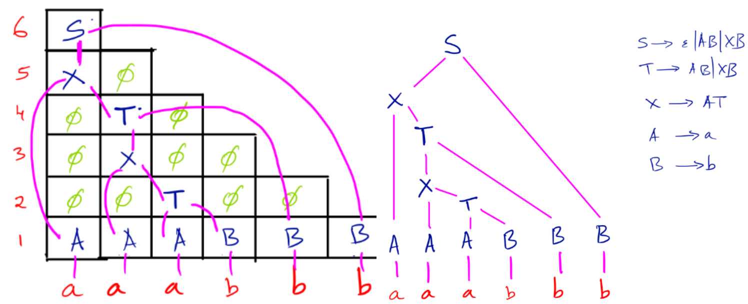
2. Grammar G is in **Chomsky Normal Form (CNF)** if all rules/productions have the form:
1. $A \to BC$ $\quad A,B,C \in V$
2. $A \to a$ $\quad a \in \Sigma$
Where $B, C \neq S$ (start var); also $S \to \epsilon$ is permitted.

3. **Theorem**: there is an algorithm which given a CFG G in CNF and string $w \in \Sigma^*$ determines whether $w \in L(G)$ in time $O(|G| \times |w|^3)$.

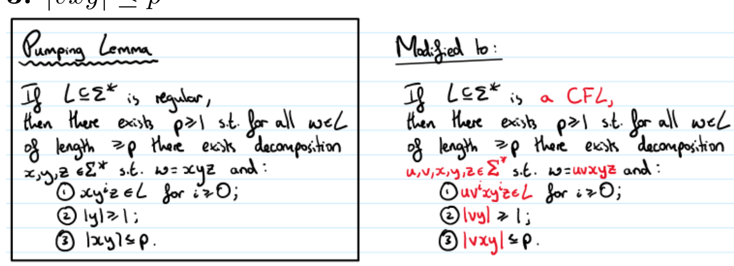
4. **Cocke-Younger-Kasami (CYK)** bottom up parsing algorithm for CFG $G = (V,\Sigma,R,S)$ **in CNF**:

CYK algorithm $\quad O(|w|^3 \times |G|)$

```
LHS(P × Q)={J ∈ V |
        G has a rule J → XY where X ∈ P and Y ∈ Q}
for i=2,..,l-1: # l is length of w
  for j=1,..l-(i-1):
    for p=1,..i-1:
      M[i,j] = M[i,j] ∪ LHS(M[p,j] × M[i-p,j+p])
w can be derived from G iff M[1,1] contains S
```



5. **CFL Pumping Lemma**: Let L be a CFL. Then $\exists$ **pumping length** $p \in \mathbb{Z}^+$ s.t. $\forall w \in L$ with $|w| \geq p$ can be decomposed as $w = uvxyz \in \Sigma^*$, where
1. $uv^ixy^iz \in L$ for all $i \geq 0$
2. $|vy| \geq 1$
3. $|vxy| \leq p$



Proof:
- Given the smallest parse tree of $w \in L(G)$, no path from root to a leaf may repeat a non-terminal, and such a path has $\leq |V|$ edges.
- If all rules have $\leq b$ symbols on the right, then such tree yields $|w| \leq b^{|V|}$.

Don't forget about **cases**!
e.g. $L = \{a^nb^nc^n : n \geq 0\}$: consider cases: 1.$vwx \subseteq a^p$, 2.$vwx \subseteq b^p$, 3. $vwx \subseteq c^p$, 4. $vwx \subseteq a^pb^p$, 5. $vwx \subseteq b^pc^p$

6. Family of CFL is closed under Union $\cup$, Kleene $*$, and concatenation, but NOT intersection $\cap$ or complement.

   Union: $L_1 = L(G_1) \cup L_2 = L(G_2)$. Assume $V_1 \cap V_2 = \varnothing$, where V_i is set of variables in G_i. Take fresh $S \notin V_1 \cup V_2 \cup \Sigma$. Set up G with S_i on start nonterminal of G_i:
   $G = (V_1 \cup V_2 \cup \{S\}, \Sigma, R_1 \cup R_2 \cup \{S \to S_1\} \cup \{S \to S_2\}, S)$

   Not closed under **Intersection**:
   $\left.\begin{array}{l} L_1 = \{a^i b^i c^k : i = j\} \\ L_2 = \{a^i b^i c^k : j = k\} \end{array}\right\}$ CFLs

   $L_1 \cap L_2 = \{a^n b^n c^n : n \geq 0\}$ not a CFL!
   $L : S \to AB, A \to aAb|\epsilon, B \to cB|\epsilon$ not a CFL

7. **Theorem**: $\exists$ CFL whose complement is not a CFL.
   Proof: otherwise $(\overline{\overline{L_1} \cup \overline{L_2}}) \to L_1 \cap L_2$ is regular.

8. **Theorem**: if $R \subseteq \Sigma^*$ os regular and $L \subseteq \Sigma^*$ is CFL, then $L \cap R$ is CFL.
   Proof:
   Let P be a PDA: $L(P) = L$.
   Let D be a DFA: $L(D) = R$.
   Construct product automaton A (also a PDA):
   - state set $Q_P \times Q_D$
   - initial state (i_P, i_D)
   - final states $F_P \times F_D$
   - stack alphabet Γ_P
   - transitions:

   $(q', \gamma') \in \delta_P(q, a, \gamma)$ where $a \neq \epsilon$, and $\delta_D(r, a) = r'$.
   Now have: $((q', r')\gamma') \in \delta_A((q, r), a, \gamma)$ for all states r of D
   Basically run PDA and DFA in tandem.

9. **Theorem**: Every **unary** CFL $L \subseteq \{a\}^*$ is regular.

10. **Commutative image** counts instances of all elements of alphabet in an input string.

11. **Theorem**: For every CFL $L \subseteq \Sigma^*$, there is a regular $R \subseteq \Sigma^*$ with commutative image $\Psi(L) = \Psi(R)$.

12. **Theorem**: If CFG G contains **no** strings of length longer than the pumping length p, then the language is finite.

    If G contains **even one** string of length longer than p, then the language is infinite.

13. **Theorem**: If CFG G contains **even one** string of length longer than pumping length p, then it also contains a string of length at most $2p - 1$

14.

Intersection	Reg.	CFL	Decidable	r.e.
Reg.	Reg			
CFL	CFL	Dec		
Decidable	Dec	Dec	Dec	
r.e.	r.e.	r.e.	r.e.	r.e.

Turing Machines

1. **Turing machine** $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ with:
   1. Q: finite set of states,
   2. Σ: finite inp alphabet without blank $\sqcup$ or start $\vdash$,
   3. Γ is tape alphabet, with $\sqcup, \vdash \in \Gamma$ and $\Sigma \subseteq \Gamma$,
   4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$: rewrite, move left/right,
   5. $q_0 \in Q$ is the start state,
   6. q_{accept} is the accept state,
   7. $q_{\text{reject}} \neq q_{\text{accept}}$ is the reject state;
   Rewrite first, then move. Immediately halt upon entering q_{acc}, q_{rej}. Have start symbol $\vdash \in \Gamma, \vdash \notin \Sigma$.

2. **Configuration** is a snapshot of what the TM looks like at any point (state, tape contents, reading head position): $X = (u, q, v) \subseteq \Gamma^* \times Q \times \Gamma^*$: tape followed by states followed by tape again $(\vdash, 0, q, 11, \sqcup)$.

3. **Start configuration**: $(\vdash, q_0, w \in \Sigma^*)$.
   $\left.\begin{array}{l} \textbf{Accepting configuration}: (u, q_{\text{accept}}, v). \\ \textbf{Rejecting configuration}: (u, q_{\text{reject}}, v). \end{array}\right\}$ **Halting**

4. Configuration (s_1, q_1, t_1) **yields** (s_1, q_2, t_2) if:
   1. (s_1, q_1, t_1) is not halting, so can proceed,
   2. if $t_1 \neq \epsilon$ and $t_1 \in a\Gamma^*$ where $a \in \Gamma$, either:
   - $\delta(q_1, a) = (q_2, b, R)$ and $s_2 = s_1 b$ and $t_1 = a t_2$; or
   - $\delta(q_1, a) = (q_2, b, L)$ and:
     - $\gg$ assuming $s_1 \neq \epsilon$ have: $s_1 = s_2 c$ for some $c \in \Gamma$, and $t_2 = cbt'$ where $t_1 = at'$;
     - $\gg$ assuming $s_1 = \epsilon$, have $s_2 = \epsilon$, $t_2 = bt'$, $t_1 = at'$ for some $t' \in \Gamma^*$

   3. if $t_2 = \epsilon$, and either:
   - $\delta(q_1 \sqcup) = (q_2, b, L)$ and $s_1 = s_2 c$ for some $c \in \Gamma$, $t_2 = cb$
   - $\delta(q_1, \sqcup) = (q_2, b, R)$ and $s_2 = s_1, b$ and $t_2 = \epsilon$.

   E.g. $abq_i cd$ yields $abc' q_j d$ if $\delta(q_i, c) = (q_j, c', R)$

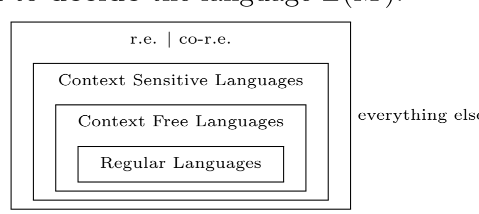
5. A **run** of TM M on input $w \in \Sigma^*$ is a finite sequence of configurations $c_0, c_1, ..c_n$ s.t.
   1. c_0 is the start config of M on w;
   2. for each $i = 1, ..n$: c_{i-1} yields c_i.

6. **Accepting/Rejecting run** if it ends in acc/rej config.

7. TM M **accepts/rejects** input $w \in \Sigma^*$ if $\exists$ acc/rej run of M on w. M **halts** on $w \in \Sigma^*$ if it accepts or rejects w.

8. Language **recognised** by TM M is
   $L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$

9. TM M is a **decider** if it rejects all strings from $\Sigma^* \setminus L(M)$.
   M is said to **decide** the language $L(M)$.



Variands of TMs, Decision Problems

1. **Stay Put** TM: $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$ where S is "do nothing", or "stay put".

2. **Bi-infinite** TM: tape has infinite $\sqcup$ to both sides of the input; but can split in half, creating two-row normal TM, each column sharing a reading head unless storing last position reference per reading head somewhere, so not more expressive power. $\delta : Q \times \Gamma^2 \to Q \times \Gamma^2 \times \{L, R\}^2$

3. **Multi-tape** TM has n tapes, each with a different reading head: $\delta : Q \times \Gamma^n \to Q \times \Gamma^n \times \{L, R\}^n$.

4. $L \subseteq \Sigma^*$ is **Turing-recognisable** or ***recursively enumerable*** **(r.e.)** if there is a TM $M : L(M) = L$.

5. $L \subseteq \Sigma^*$ is **Turing-decidable** or ***recursive*** if there is a decider $D : L(D) = L$ (never halts).

6. For any object O (e.g. TM, PDA, NFA, DFA, etc): write $\langle O \rangle$ for the **encoding** of O as string over appropriate Σ. Also, $\langle O, w \rangle \in \Sigma_0^*$: single string encoding of O and $w \in \Sigma$ pair over alphabet Σ (basically a file).

7. **Acceptance Decision Problem for PDA**:
- input: PDA $\mathcal{A}$, input string $w \in \Sigma^*$.
- output: does $\mathcal{A}$ accept w?
$\mathcal{A}_{\text{PDA}} = \{\langle \mathcal{A}, w \rangle : \mathcal{A} \text{ is a PDA}, \mathcal{A} \text{ accepts } w\}$.
$\mathcal{A}_{\text{PDA}}$ is decidable

8. **Acceptance Decision Problem for TM**:
$\mathcal{A}_{TM} = \{\langle M, w \rangle : M \text{ is a TM that accepts } w\}$
Theorem: $\mathcal{A}_{TM}$ is undecidable.
Proof by contradiction:
Assume FTSOC $\mathcal{A}$ is a decider for $\mathcal{A}_{TM}$. Then on input $\langle M, w \rangle$:
   - $\mathcal{A}$ accepts if M accepts w, $\mathcal{A}$ rejects if M doesn't accept w.

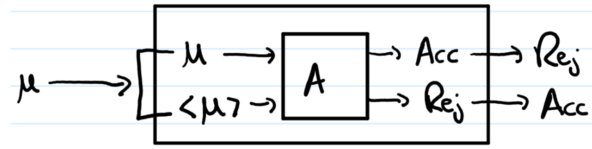
   Let D be a new TM, which on input $\langle M \rangle$ (where M is a TM), simulates $\mathcal{A}$ on $\langle M, \langle M \rangle \rangle$ (i.e. $w = \langle M \rangle$),
   - if $\mathcal{A}$ accepts (M accepts $\langle M \rangle$), D rejects,
   - but if A rejects (if M rejects $\langle M \rangle$), D accepts.

   If D accepts $\langle D \rangle$, it rejects $\langle D \rangle$. If D rejects $\langle D \rangle$, it accepts $\langle D \rangle$

Decidability

1. **Theorem** $\mathcal{A}_{TM}$ is recursively enumerable & undecidable.
Proof: take TM U (**interpreter**) which on input $\langle M, w \rangle$:
1. Simulates execution of code M on input w step by step.
2. if M accepts/rejects so does U. $\square$

2. If $\mathcal{A}$ is a decider for $\mathcal{A}_{TM}$ then $\mathcal{A}$ takes as input code M and input w and either returns accept or rejects. Use this as a black box to build decider $\mathcal{D}$, that accepts M then inside feeds machine M and description of machine $\langle M \rangle$.



3. **Theorem** Class of decidable languages is closed under union, intersection, complement and kleene star.

   1) $w \to$ [M] $\to$ accept, $\to$ reject   2) $w \to$ [[M] $\to$ accept, $\to$ reject] $\to$ rej, $\to$ acc

   Proof: suppose L is decidable, let 1) TM M be a decider for L, $L = L(M)$; and 2) TM M' be its complement.
   - **Intersection** - accept if both (1,2) accept.
   - **Union** - accept if any of the two (1 or 2) deciders accept.
   - **Kleene star**: there is a decider for L^* where:
     1. Given input $w \in \Sigma^*$, consider all partitions of w into substrings. For each part, run M.
     2. If for some partitioning M accepts every part, then accept, otherwise reject.

4. Suppose Σ, Δ are finite alphabets. Function $f : \Sigma^* \to \Delta^*$ is **Turing Computable** if $\exists$ decider that $\forall w \in \Sigma^*$ halts leaving $f(w)$ on the tape.

5. **Church-Turing thesis**: anything that can be described algorithmically has a TM.

6. **Theorem**: Let L be language over alphabet Σ^*. L is decidable iff both L and complement $\Sigma^* \setminus L$ are r.e.
Proof: ($\Rightarrow$): Suppose $L = L(D)$, D is decider, since D is a TM, then L is r.e. To recognise $L = \Sigma^* \setminus L$ run D and flip the answer.
($\Leftarrow$): let M_1, M_2 be TMs for L and $\overline{L}$ respectively. Given input $w \in \Sigma^*$, run M_1 on w and, in parallel, run M_2 on w, then one of M_1, M_2 must halt since if $w \in L$ then M_1 eventually accepts so accept, if not - then M_2 does, so reject. Hence this is a decider for L.

Halting

1. L decidable $\equiv L, \overline{L}$ are turing-recognisable.

2. **Corollary**: $\overline{\mathcal{A}_{TM}} = \{\langle M, w \rangle$: M is a TM, string w and M does not accept $w\}$ is not r.e. (infinite loops aren't turing recognisable).

3. **Co-recursively enumerable (co-r.e.)**: languages whos complement is r.e.

4. **Intersection** of r.e. and co-r.e. is decidable.

5. There are countably many TMs, but uncountably many languages $L \subseteq \{a, b\}^*$, so not all languages are decidable.

6. **Halting problem** $\text{HALT}_{TM} = \{\langle M, w \rangle : M \text{ is a TM}, w \text{ is a string and } M \text{ halts on input } w\}$.

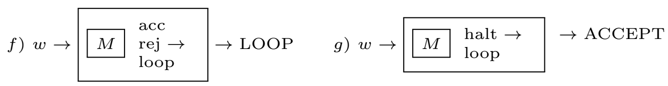
7. Suppose A, B are languages where $A \subseteq \Sigma^*, B \subseteq \Delta^*$. Then Δ is **reducible** to B if there is a computable function $f : \Sigma^* \to \Delta^*$ s.t. $\forall w \in \Sigma^* : w \in A$ iff $f(w) \in B$.

8. Write $A \ll_m B$, f is a **many-one** reduction (1 call).

9. **Lemma**: if $A \leq_m B$ and B decidable then A is decidable.
Proof: Let f be a reduction from A to B, so R is the TM that on input $w \in \Sigma^*$ produces $f(w)$.
Let M be a decider for B. Use R to decide if $f(w)$ is in B, then accept if so, reject otherwise. □

10. $\mathcal{A}_{TM} \leq_m \text{HALT}_{TM} \leq_m \mathcal{A}_{TM}$, and $\mathcal{A}_{TM}$ is undecidable implies that **HALT**$_{TM}$ **is undecidable**.

Proof: for $A_{TM} \leq_m^f \text{HALT}_{TM} \leq_m^g A_{TM}$:



(⇒) On input $\langle M, w\rangle$, f produces $\langle M', w\rangle$.
Now, M' : on input w run M on w, if it accepts then accept, if rejects then loop forever.

(⇐) On input $\langle M, w\rangle$, g produces $\langle M', w\rangle$.
Now, M': on input w, run M on w; if it accepts or rejects, then accept.

11. Given a TM M, **all of** $1, 2, 3$ **are undecidable for TM**
Does M halt on **1.** ϵ? **2.** $L(M) \neq \varnothing$? **3.** Is $L(M) = \Sigma^*$?

Proof for $L_\epsilon = \{\langle M\rangle : M$ is a TM and T halts on $\epsilon\}$ by reduction $A_{TM} \leq_m \text{HALT}_\epsilon$
• To solve acceptance problem on input $\langle M, w\rangle$: run M on w: if M accepts, accept, if M rejects - loop forever.
• Computable f (reduction $\mathcal{A}_{TM} \leq_m^f$) on input $\langle M, w\rangle$, where M is a TM and w is a string: Construct TM $N_{M,w}$, which given input string x **ignores** x and runs M on w - accepting if M accepts and loops forever if M rejects.

• The output of reduction $f(\langle M, w\rangle) = \langle N_{M,w}\rangle$.
• For all $s \in \Sigma^* : s \in A$ iff $f(s) \in B$.
• For all $\langle M, w\rangle$, M accepts w iff $N_{M,w}$ halts on ϵ.

If M accepts w, then $N_{M,w}$ halts on ϵ.
If M doesn't accept w, then $N_{M,w}$ doesn't halt on ϵ.

(2), (3) follow the same logic - if M doesn't accept anything then $L(M) = \varnothing$, if accepts everything - $L(M) = \Sigma^*$.

Basically take a machine that does something, give it your custom input, and see if it can arrive at some answer with it ignoring all other inputs other than yours - if it does, then you can accept.

12. **Rice Theorem**: all non-trivial (not $\top/\bot$) semantic properties of programs are undecidable.

13. **Theorem**: Recognisable (r.e.) languages are closed under: $\cap, \cup, *$ but not complement.

Proof: Problem P is decidable if both P and $\overline{P}$ are r.e. We know that H_{TM} is undecidable and r.e., therefore $\overline{H_{TM}}$ is not r.e. So not closed under complement.

14. **Enumerator** is a variant of a multi-tape TM with:
- work tape (read-write), output tape (read-only),
- distinguishable "enum" state.

Initially, both tapes empty (always runs on empty input). When "enum" is reached, flush the output; continue.

Enumerates the strings it outputs, hence its language, or the set of strings it produces is **exactly r.e.**

15. If $A \leq_m B$ and B decidable, then A decidable.

16. If $A \leq_m B$ and A undecidable, then B undecidable.

17. Problem P is decidable if both P and $\overline{P}$ are r.e.

18. If $A \leq_m B$ and B is r.e., then A is r.e.

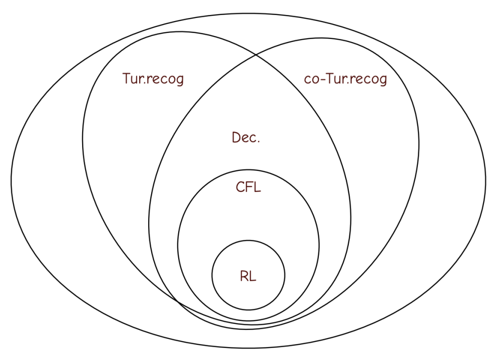
19.

	Reg.	**CFL**	**Decidable**	**r.e.**
Complement	Y	N	Y	N
Union ∪	Y	Y	Y	Y
Intersection ∩	Y	N	Y	Y
Kleene ∗	Y	Y	Y	Y
Concatenation	Y	Y	Y	Y

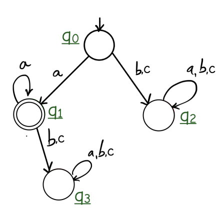
20.

Regular	**CFL**	**R.E languages**
NFA/DFA/GNFA	PDA	TM
Regular Grammars	CFG	Type 0 grammars
Pushdown, Myhil-Nerode	CFL Pushdown	Reductions

21.



22. State Diagram and State transition table.



STATE DIAGRAM

δ	a	b	c
→ q0	q1	q2	q2
* q1	q1	q3	q3
q2	q2	q2	q2
q3	q3	q3	q3

STATE TRANSITION TABLE