CS4232
Theory of Computation

Preliminaries

- $0 \in \mathbb{N}$
- **Alphabet** (Σ): finite (non-empty) set of symbols
- **String**: finite sequence of symbols from a given alphabet
  - empty string: ε or Λ
- **Language** (L): a set of strings (over an alphabet)
  - $L_1 \cdot L_2 = L_1 L_2 \coloneqq \{xy \mid x \in L_1, y \in L_2\}$
  - $L^* \coloneqq \{x_1 \dots x_n \mid x_1, \dots, x_n \in L, n \in \mathbb{N}\}$
  - $L^+ \coloneqq \{x_1 \dots x_n \mid x_1, \dots, x_n \in L, n \geq 1\}$
  - Num of strings over any fixed finite alphabet is countable
  - Num of lang. over any non-empty alphabet is uncountable

Regular Languages

- DFA, NFA, ε-NFA, Regex are all equivalent (in terms of the set of expressible languages)

Deterministic Finite Automata (DFA)

- $A \coloneqq (Q, \Sigma, \delta, q_0, F)$, where:
  - Q is a finite set of states
  - Σ is a (finite) alphabet
  - $\delta : Q \times \Sigma \to Q$ is a function
  - $q_0 \in Q$ is the starting state
  - $F \subseteq Q$ is the set of final states
- **Transition table** (example):

	0	1
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_2

- **Transition function for strings**:
  - $\hat{\delta}(q, \varepsilon) \coloneqq q$
  - $\hat{\delta}(q, xa) \coloneqq \delta(\hat{\delta}(q, x), a)$
- **Language accepted**:
  $L(A) = Lang(A) \coloneqq \left\{ w \mid \hat{\delta}(q_0, w) \in F \right\}$
  (i.e. whether applying each char terminates in F)
- **Dead state** q: $\forall w \in \Sigma^*, \hat{\delta}(q, w) \notin F$
  (i.e. cannot reach any final state from q)
- **Unreachable state** q: $\forall w \in \Sigma^*, \hat{\delta}(q_0, w) \neq q$
  (i.e. cannot reach q from q_0)

Nondeterministic Finite Automata (NFA)

- $A \coloneqq (Q, \Sigma, \delta, q_0, F)$, where:
  - Q is a finite set of states
  - Σ is a (finite) alphabet
  - $\delta : Q \times \Sigma \to 2^Q$ is a function
  - $q_0 \in Q$ is the starting state
  - $F \subseteq Q$ is the set of final states
- **Transition table** (example):

	ε	0	1
q_0	$\{q_1, q_2\}$	$\{q_0\}$	...
q_1	...	...	...
q_2	...	...	...

- **Transition function for strings**:
  - $\hat{\delta}(q, \varepsilon) \coloneqq \{q\}$
  - $\hat{\delta}(q, xa) \coloneqq \bigcup_{p \in \hat{\delta}(q,x)} \delta(p, a)$
- **Language accepted**:
  $L(A) = Lang(A) \coloneqq \left\{ w \mid \hat{\delta}(q_0, w) \cap F \neq \varnothing \right\}$
  (i.e. whether there is a path of chars that terminates in F)
- To show that every language acceptable by NFA is also acceptable by some DFA:
  Given NFA $A \coloneqq (Q, \Sigma, \delta, q_0, F)$,
  define DFA $A_D \coloneqq (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$, where
  - $Q_D \coloneqq 2^Q$
  - $F_D \coloneqq \{S \mid S \subseteq Q \text{ and } S \cap F \neq \varnothing\} \subseteq Q_D$
  - $\delta_D(S, a) = \bigcup_{q \in S} \delta(q, a)$
  and then prove that for any string w, $\hat{\delta_D}(\{q_0\}, w) = \hat{\delta}(q_0, w)$ by induction on length of w
- When simulating the NFA to DFA algorithm, omit unreachable states and follow it like a flood-fill:

	0	1
$\{q_0\}$	$\{q_0, q_1\}$	...
$\{q_0, q_1\}$	...	...
...	...	...

NFA with ε transitions (ε-NFA)

- $A \coloneqq (Q, \Sigma, \delta, q_0, F)$, like NFA, but $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$
- ε **closure**: $Eclose : Q \to 2^Q$ is defined recursively:
  - $q \in Eclose(q)$
  - $p \in Eclose(q) \implies \forall p' \in \delta(p, \varepsilon),\ p' \in Eclose(q)$
  (note: we define $Eclose$(<set>) to return a set union)
- **Transition function for strings**:
  - $\hat{\delta}(q, \varepsilon) \coloneqq Eclose(q)$
  - $\hat{\delta}(q, wa) \coloneqq \bigcup_{p \in R} Eclose(p)$ where $R = \bigcup_{p \in \hat{\delta}(q,w)} \delta(p, a)$
- To show that every language acceptable by ε-NFA is also acceptable by some DFA:
  Given ε-NFA $A \coloneqq (Q, \Sigma, \delta, q_0, F)$,
  define DFA $A_D \coloneqq (Q_D, \Sigma, \delta_D, Eclose(q_0), F_D)$, where
  - $Q_D \coloneqq 2^Q$
  - $F_D \coloneqq \{S \mid S \subseteq Q \text{ and } S \cap F \neq \varnothing\} \subseteq Q_D$
  - $\delta_D(S, a) \coloneqq \bigcup_{p \in R} Eclose(p)$ where $R = \bigcup_{p \in S} \delta(p, a)$

Regular Expressions

- Defined recursively:
  - $L(\varepsilon) \coloneqq \{\varepsilon\}$
  - $L(\varnothing) \coloneqq \varnothing$
  - $a \in \Sigma \implies L(a) \coloneqq \{a\}$
  - $L(r_1 + r_2) \coloneqq L(r_1) \cup L(r_2)$
  - $L(r_1 \cdot r_2) \coloneqq \{xy \mid x \in L(r_1) \text{ and } y \in L(r_2)\}$
  - $L(r_1^*) \coloneqq \{x_1 \cdots x_k \mid k \in \mathbb{N} \text{ and } x_i \in L(r_1)\ \forall 1 \leq i \leq k\}$
  - $L((r_1)) \coloneqq L(r_1)$
- To show that every language acceptable by DFA is also accepted by some regular expression:
  - Given DFA $A \coloneqq (Q, \Sigma, \delta, q_{start}, F)$ where $Q = \{1, \dots, n\}$ for some $n \in \mathbb{N}$ and $q_{start} = 1$,
  let $R^k_{i,j}$ be the regular expression for the set of strings formable by going from state i to state j using intermediate states numbered $\leq k$ (note: i and j may be more than k), and prove by induction on k
  - $R^0_{i,j} = \begin{cases} \sum_r a_r & \text{if } i \neq j \\ \varepsilon + \sum_r a_r & \text{if } i = j \end{cases}$ where $\delta(i, a_r) = j$
  - $R^{k+1}_{i,j} = R^k_{i,j} + R^k_{i,k+1} \left(R^k_{k+1,k+1} \right)^* R^k_{k+1,j}$
  - Then a regular expression for $L(A)$ is $\sum_{j \in F} R^n_{1,j}$
- To show that every language acceptable by regular expression is also accepted by some ε-NFA:
  - (the ε-NFA built additionally satisfies: only one final state, no transition into starting state, no transition out of final state, the starting and final states are different)
  - base cases: $\varnothing$, ε, and a regular expressions: two states; set edge as appropriate
  - induction case: consider how to combine the ε-NFA for $r_1 + r_2$, $r_1 \cdot r_2$, r_1^* (remember to add εs)
- **Identities and extensions** (the languages accepted are equivalent):
  - $M + N = N + M$
  - $L(M + N) = LM + LN$
  - $L + L = L$
  - $(L^*)^* = L^*$
  - $\varnothing^* = \varepsilon$
  - $\varepsilon^* = \varepsilon$
  - $L^+ = LL^* = L^*L$
  - $L^* = \varepsilon + L^+$
  - $(L + M)^* = (L^* M^*)^*$

DFA Minimisation

- **Equivalence classes** of strings in a language:
  $u \equiv_L v \ \coloneqq\ \forall x, ux \in L \iff wx \in L$
- Given a regular language L over Σ, the equivalence classes (if finite) form a unique minimal DFA $(Q, \Sigma, \delta, q_0, F)$, where:
  - $Q \coloneqq \{equiv(w) \mid w \in \Sigma^*\}$
  - $\delta(equiv(w), a) \coloneqq equiv(wa)$ (this is well-defined)
  - $q_0 \coloneqq equiv(\varepsilon)$
  - $F \coloneqq \{equiv(w) \mid w \in L\}$
- States (p, q) (unordered pair) are **distinguishable**: $\exists$ string w such that exactly one of $\hat{\delta}(p, w)$ and $\hat{\delta}(q, w)$ is in F
  (can be shown that a distinguishable pair must be distinguishable by a suffix no longer than n^2 length)
- **Table building algorithm** to determine all distinguishable pairs:
  - Base case: each pair (p, q) such that $p \in F$ and $q \notin F$ (or vice versa) is distinguishable
  - Inductive step: for any $a \in \Sigma$, if $(\delta(p, a), \delta(q, a))$ is distinguishable, then (p, q) is distinguishable
  Example ("X1": final vs non-final; "X2": distinguishable by X1, ...):

q_1					
q_2	X3	X3			
q_3	X3	X3			
q_4	X1	X1	X1	X1	
q_5	X2	X2	X2	X2	X1
	q_0	q_1	q_2	q_3	q_4

- **DFA minimisation algorithm**:
  0. Delete all non-reachable states
  1. Find all nondistinguishable pairs of states (they give an equivalence relation)
  2. Build the new automata $(Q, \Sigma, \delta, q_0, F)$, where:
  - Q is the set of equivalence classes
  - $\delta(equiv(p), a) \coloneqq \delta(equiv(q))$ where $\delta_{orig}(p, a) = q$
  - q_0 is the equivalence class of the original starting state
  - F is the set of equivalence classes containing a final state (all such equivalence classes will only contain final states)

Regular Languages

- **Pumping lemma**: If L is a regular language, then there exists some $n > 0$ such that $\forall w \in L$ where $|w| \geq n$, we can break w into three strings $w = xyz$ such that:
  - $y \neq \varepsilon$
  - $|xy| \leq n$
  - $\forall k \geq 0,\ xy^k z \in L$
  Proof: Let n be the number of states in a DFA that accepts L, and consider the $(n+1)$ prefixes of w of lengths $0, \dots, n$, and apply Pigeonhole principle on the states reached by those $(n+1)$ prefixes
  Corollary: All finite languages (languages containing a finite number of strings) are regular
- **Closure properties**: If L_1 and L_2 are regular languages, then the following are regular:
  - $L_1 \cup L_2$
  - $L_1 \cdot L_2$
  - $L_1 \cap L_2$
  - $L_1 - L_2$
  If L is a regular language, then the following are regular:
  - $\overline{L} \coloneqq \Sigma^* - L$
  - L^R (the set formed by reversing every string in L)
- **Homomorphism**: $h : \Sigma \to B^*$, where Σ, B are alphabets
  - For string, define $h : \Sigma^* \to B^* : a_1 \cdots a_n \mapsto h(a_1) \cdots h(a_n)$
  - If L is regular, then $h(L)$ is also regular
- **Parallel simulation**: Taking the product of both sets of states; choice of δ and F depends on the problem; can easily model union and intersection of regular languages

Context-Free Languages

Context-Free Grammars

- $G \coloneqq (V, T, P, S)$, where:
  - V is a finite set of variables (aka. non-terminals)
  - T is a finite set of terminals
  - P is a finite set of productions of the form $A \to \gamma$, where $A \in V$ and $\gamma \in (V \cup T)^*$
  - $S \in V$ is the start symbol (note: S can be implicitly start)
- **Derivations**: $\alpha A \beta \Rightarrow_G \alpha \gamma \beta$: there is a production $A \to \gamma$
  $\alpha \Rightarrow^*_G \beta$ is defined inductively:
  - $\alpha \Rightarrow^*_G \alpha$ for all $\alpha \in (V \cup T)^*$
  - If $\alpha \Rightarrow^*_G \beta$ and $\beta \Rightarrow_G \gamma$, then $\alpha \Rightarrow^*_G \gamma$
- **Language accepted**: $L(G) \coloneqq \{w \in T^* \mid S \Rightarrow^*_G w\}$
- **Sentential form**: Any α such that $S \Rightarrow^*_G \alpha$
- **Left-most derivation**: replace left-most non-terminal
  Right-most derivation: replace right-most non-terminal
  - There is exactly one left-most (resp. right-most) derivation for each valid parse tree
- **Right-linear grammar**: G is right-linear if all productions are in one of these forms:
  - $A \to wB$, where $w \in T^*$ and $B \in V$
  - $A \to w$, where $w \in T^*$
  Thm: Right-linear grammar is equiv. to regular language

- To show that every language acceptable by DFA is also generated by some right-linear grammar:
  - Given DFA $A := (Q, \Sigma, \delta, q_0, F)$ (WLOG assume $Q \cap \Sigma = \varnothing$), define $G := (Q, \Sigma, P, q_0)$, where:
  - $\forall q, p \in Q$, $\forall a \in \Sigma$, if $\delta(q, a) = p$ then add rule $q \to ap$ to P
  - $\forall q \in F$, add rule $q \to \varepsilon$ to P
  and then prove that for any string w,
  $\hat{\delta}(q_0, w) = p \iff q_0 \Rightarrow^*_G wp$
  (and hence $\hat{\delta}(q_0, w) \in F \iff q_0 \Rightarrow^*_G w$)

- To show that every language generated by some right-linear grammar is also acceptable by some ε-NFA:
  - Given $G := (V, \Sigma, P, S)$ (WLOG assume $V \cap \Sigma = \varnothing$, and each production is in the form $A \to bC$ or $A \to \varepsilon$ where $b \in \Sigma \cup \{\varepsilon\}$ and $A, C \in V$ (we can split up production rules if this is not already the case)), define ε-NFA $A := (V, \Sigma, \delta, S, F)$, where
  - $F := \{A \mid A \to \varepsilon$ is a production in $P\}$
  - if $A \to aB$ is a production in P, then $B \in \delta(A, a)$
  and then prove that $A \Rightarrow^*_G wB \iff B \in \hat{\delta}(A, w)$
  (and hence $A \Rightarrow^*_G w \iff \hat{\delta}(A, w) \cap F \neq \varnothing$)
  (and hence $S \Rightarrow^*_G w \iff \hat{\delta}(S, w) \cap F \neq \varnothing$)

- **Inherently ambiguous language**: Any CFG for it will have ambiguous parse trees

Pushdown Automata (PDA)

- $P := (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, where:
  - Q is a finite set of states
  - $q_0 \in Q$ is the start state
  - $F \subseteq Q$ is the set of final states
  - Σ is the input alphabet
  - Γ is the stack alphabet
  - $Z_0 \in \Gamma$ is the initial stack symbol
  - $\delta : Q \times (\Sigma \cup \varepsilon) \times \Gamma \to 2^{Q \times \Gamma^*}$ is a function
  (where $(p, \gamma) \in \delta(q, a, X)$ means that when in state q, reading symbol a, top of stack being X, then the machine's new state is p, the X at top of stack is popped, and γ is pushed to the stack (right side goes into stack first))

- **Instantaneous descriptions**:
  (q, w, α) means the current state is q, the input left is w, and α is the current stack state
  - $(q, aw, X\alpha) \vdash (p, w, \beta\alpha) := (p, \beta) \in \delta(q, a, X) \quad (a$ can be $\varepsilon)$
  - $I \vdash^* J \quad := \quad I = J \quad$ or $\quad (I \vdash^* K$ and $K \vdash J)$

- **Possible acceptance conditions**: (they are equivalent)
  By final state: $\{w \mid \exists q_f \in F$ such that $(q_0, w, Z_0) \vdash^*_P (q_f, \varepsilon, \alpha)\}$
  By empty stack: $\{w \mid \exists q \in Q$ such that $(q_0, w, Z_0) \vdash^*_P (q, \varepsilon, \varepsilon)\}$

- **Acc. by empty stack $\implies$ Acc. by final state**:
  (intuitively: initially add a special stack symbol, and if that special symbol is encountered then go to the final state)
  Given $P := (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, then
  let $P_F := (Q \cup \{p_0, p_f\}, \Sigma, \Gamma \cup \{X_0\}, \delta_F, p_0, X_0, \{p_f\})$, where
  - $\delta_F(p_0, \varepsilon, X_0) = \{(q_0, Z_0X_0)\}$
  - $\forall p \in Q, \forall a \in \Sigma \cup \{\varepsilon\}, \forall Z \in \Gamma$,
    $\delta_F(p, a, Z)$ contains all $(q, \gamma) \in \delta(p, a, Z)$
  - $\forall p \in Q, \delta_F(p, \varepsilon, X_0)$ contains (p_f, ε)

- **Acc. by final state $\implies$ Acc. by empty stack**:
  (intuitively: from every final state, add a transition to a special state p_f that empties the stack (note: still need special stack symbol because the existing PDA might empty the stack in a non-final state))
  Given $P := (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, then
  let $P_E := (Q \cup \{p_0, p_f\}, \Sigma, \Gamma \cup \{X_0\}, \delta_E, p_0, X_0, \{p_f\})$, where
  - $\delta_E(p_0, \varepsilon, X_0) = \{(q_0, Z_0X_0)\}$
  - $\forall p \in Q, \forall a \in \Sigma \cup \{\varepsilon\}, \forall Z \in \Gamma$,
    $\delta_E(p, a, Z)$ contains all $(q, \gamma) \in \delta(p, a, Z)$
  - $\forall p \in F, \forall Z \in \Gamma \cup \{X_0\}, \delta_E(p, \varepsilon, Z)$ contains (p_f, ε)
  - $\forall Z \in \Gamma \cup \{X_0\}, \delta_E(p_f, \varepsilon, Z)$ contains (p_f, ε)

- Note: in above two constructions, the constructed PDA works both for final state and empty stack models

Equivalence of CFGs and PDAs

- To show that every CFG is accepted by a PDA (empty stack model):
  (intuitively, use left-most derivation and use stack to keep track of "what is left to derive")
  Given $G := (V, T, P, S)$, then
  let $PDA := (\{q_0\}, T, V \cup T, \delta, q_0, S, F)$, where
  - $\forall a \in T, \delta(q_0, a, a) = \{(q_0, \varepsilon)\}$
  - $\forall A \in V, \delta(q_0, \varepsilon, A) = \{(q_0, \gamma) \mid (A \to \gamma) \in P\}$

- To show that every PDA (empty stack model) is accepted by a CFG:
  (intuitively, each production rule fully removes one item (and all the children that spawn) from the stack)
  Given $PDA := (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, then
  let $G := (V, \Sigma, R, S)$, where
  - $V := \{S\} \cup \{[qZp] \mid q, p \in Q, Z \in \Gamma\}$
  - $\forall p \in Q$, we have production $S \to [q_0Z_0p]$
  - If $(r, Y_1 \cdots Y_k) \in \delta(q, a, X)$, then $\forall r_1, \ldots, r_k \in Q$, we have production $[qXr_k] \to a[rY_1r_1][r_1Y_2r_2] \cdots [r_{k-1}Y_kr_k]$

Deterministic PDA

- PDA where both conditions are satisfied:
  - $\forall a \in \Sigma \cup \{\varepsilon\}, \forall Z \in \Gamma, \forall q \in Q$, there is at most one element on $\delta(q, a, Z)$
  - if $\delta(q, \varepsilon, X)$ is non-empty, then $\delta(q, a, X)$ is empty for all $a \in \Sigma$

- **Thm**: There exists a language which is accepted by PDA but not by any DPDA

- Every regular language can be accepted by DPDA with final state
  - just don't use the stack

- DPDA with empty stack cannot accept some regular langauges
  - if $w \in L$ then we can't accept any w' that contains w as a prefix

Chomsky Normal Form

- All productions are in these forms:
  - $A \to BC$ (where $A, B, C \in V$)
  - $A \to a$ (where $a \in T$ and $A \in V$)
  (note: no ε on purpose)

- A is **useful**: $\exists \alpha, \beta \in (V \cup T)^*, \exists w \in T^*$ s.t. $S \Rightarrow^* \alpha A \beta \Rightarrow^* w$
  A is **useless**: A is not useful

- A is **generating**: $\exists w \in T^*$ such that $A \Rightarrow^* w$
  To determine generating symbols:
  - Base case: all symbols in T are generating
  - Inductive step: if there is a production $A \to \alpha$ and α consists only of generating symbols, then A is generating

- A is **reachable**: $\exists \alpha, \beta \in (V \cup T)^*$ such that $S \Rightarrow^* \alpha A \beta$
  To determine reachable symbols:
  - Base case: S is reachable
  - Inductive step: if A is reachable and $A \to \alpha$ is a production, then all symbols in α are reachable

- A is useful $\implies$ A is generating and reachable
  (note: converse is not necessarily true)

- **Eliminating useless symbols**:
  1. Eliminate all non-generating symbols
  2. Eliminate all non-reachable symbols
  The resulting CFG does not contain any useless symbols

- A is **nullable**: $A \Rightarrow^* \varepsilon$
  To determine nullable symbols:
  - Base case: if $A \to \varepsilon$ then A is nullable
  - Inductive step: if $A \to \alpha$ and every symbol in α is nullable, then A is nullable

- **Eliminating ε productions**: Determine all nullable non-terminals and replace each production of that nonterminal $A \to \alpha$ with $A \to \alpha'$ where α' can be formed from α by possibly deleting some of the non-terminals which are nullable (but omit the production when $\alpha = \varepsilon$)
  - e.g. If A and C are nullable then convert $A \to ABaCd$ to $A \to ABaCd|BaCd|ABad|Bad$ - this method produces the language $L(G') = L(G) - \{\varepsilon\}$
  - proof by induction that $\forall A \in V, \forall w \in T^* - \{\varepsilon\}$,
  $A \Rightarrow^*_G w \iff A \Rightarrow^*_{G'} w$
  (note: if we really want nullable S, then we can wrap the nonnullable grammar with a new symbol to ε)

- **Eliminating unit productions** (i.e. determining $A \Rightarrow^* B$ for any non-terminals A and B):
  - Base case: (A, A) is a unit pair
  - Inductive step: if (A, B) is a unit pair and $B \to C$ is a production, then (A, C) is a unit pair
  Then for any unit pair (A, B), remove the unit productions of A, and for every production $B \to \gamma$ add production $A \to \gamma$

- **Eliminating overlong productions**:
  All productions of length at least 2 can be converted to acceptable form:
  Given production $A \to X_1 \cdots X_k$, replace with:
  - $A \to Z_1B_2$
  - $B_2 \to Z_2B_3$
  $\vdots$
  - $B_{k-1} \to Z_{k-1}Z_k$
  - $Z_i \to X_i$ if X_i is a terminal
  - $Z_i = X_i$ (i.e. replace Z_i with X_i in above rules) if X_i is a nonterminal

- **Thm on size of parse tree**: Suppose we have a parse tree using a Chomsky Normal Form Grammar. If the length of the longest path from root to a node is s, then the size of the string generated is at most 2^{s-1}

- **Pumping lemma**: If L is a context-free language, then there exists some $n > 0$ such that $\forall z \in L$ where $|z| \geq n$, we can break z into five strings $z = uvwxy$ such that:
  - $vx \neq \varepsilon$
  - $|vwx| \leq n$
  - $\forall i \geq 0, uv^iwx^iy \in L$
  Proof: In the CNF parse tree of any string of length at least $n = 2^m$, there is a path of length at least $m + 1$, so there must be two non-terminals which are same

- **Ogden's lemma**: If L is a context-free language, the there exists some $n > 0$ such that $\forall z \in L$ with a least n distinguished positions, we can break z into five strings $z = uvwxy$ such that:
  - vx has at least one distinguished position
  - vwx has at most n distinguished positions
  - $\forall i \geq 0, uv^iwx^iy \in L$

- **Closure properties**:
  - Union: If L_1 and L_2 are context-free, then $L_1 \cup L_2$ is context-free too
  - Substitution: If L is context-free, and given any mapping s from each terminal a to a context-free language L_a, we define s on strings as such:
  $s(\varepsilon) := \{\varepsilon\}$
  $s(wa) = s(w) \cdot s(a), \ \forall a \in \Sigma, \forall w \in \Sigma^*$
  Then $\bigcup_{w \in L} s(w)$ is context-free
  - Reversal: If L is context-free, then $L^R := \{w^R \mid w \in L\}$ is context-free
  - Context-free $\cap$ regular: If L is context-free and R is regular, then $L \cap R$ is context-free
  - Note: Intersection might not be context-free

- **Testing whether CFL is $\varnothing$**: Check whether S is a useless symbol

- **Testing membership in a CFL**: Convert to CNF, and use a dynamic programming algorithm; for $w = a_1 \cdots a_n$, we determine the set $X_{i,j}$ of non-terminals which generate the string $a_ia_{i+1} \cdots a_j$
  - Base case: $X_{i,i}$ is the set of non-terminals that generate a_i
  - Inductive step: $X_{i,j}$ is the set of all A such that $A \to BC$ and $B \in X_{i,k}, C \in X_{k+1,j}, \forall i \leq k < j$
  Then w is in the language iff $S \in X_{1,n}$
  Example:

i \ j	1	2	3	4	5	6	7	8
1	CD	A	CSB	A	CSB	A	CSB	A
2		CD	A	CB	A	CSB	A	CSB
3			BCD	A	CB	A	CSB	A
4				CD	A	CSB	A	CSB
5					CD	A	CSB	A
6						BCD	A	CB
7							BCD	A
8								CD

- **Greibach Normal Form**: All productions are of the form $A \to a\alpha$ where a is a terminal and α is a string of zero or more terminals or non-terminals
  - all context-free languages not containing ε have a Greibach Normal Form grammar

Turing Machines

- $M := (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, where:
  - Q is a finite set of states
  - $q_0 \in Q$ is the start state
  - Γ is the tape alphabet
  - $\Sigma \subseteq \Gamma$ is the input alphabet
  - $B \in \Gamma - \Sigma$ is the blank symbol
  - $F \subseteq Q$ is the set of final states
  - $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is a function

- **Instantaneous description**: $x_0x_1 \cdots x_{n-1}qx_nx_{n+1} \cdots x_m$ means that the tape state is $x_0 \cdots x_m$ (all other symbols are blanks) and the head is at position n (seems like there is no

memory of the "initial cell", so we can't calculate references from it)
- '$\vdash$': one-step state transition
- $I \vdash^* J \quad := \quad I = J \quad \text{or} \quad (I \vdash^* K \text{ and } K \vdash J)$

- **Language accepted**: $L(M) = \{x \mid q_0 x \vdash^* \alpha q_f \beta \text{ for some } q_f \in F\}$ (by convention, once we enter an accepting state, we stop and accept the input)
- **Function computed**: the content of the tape after it halts is the output of f (if it does not halt, then f is not defined for the given input)
- L is **recursively enumerable**: Some Turing machine accepts L
- L is **recursive (decidable)**: Some Turing machine accepts L, and halts on all inputs
- f is **partial recursive (partially computable)**: Some Turing machine computes f (it halts and output $f(x)$ for all x on which f is defined, and it does not halt on all other inputs)
- f is **recursive (computable)**: Some Turing machine computes f and f is defined an all elements of Σ^*
- **Halting problem**: It is not possible to determine if a Turing machine will halt on a particular input
- **Equivalent extensions**:
  - stay where you are ('S' move)
  - storage in finite control (extra memory to store finite values, equivalent to growing the state)
  - multiple tracks on a single tape
  - subroutines
  - semi-infinite tapes (i.e. tapes that are only infinite on one end)
  - multiple tapes (combine them into multiple tracks on a single tape, and add one more track per original tape to store a marker at the head position; then for one step of the original machine, we look at all the current values (stored in a finite store); time complexity is $O(t^2)$, where the original machine took t time)
  - non-deterministic Turing machines ($\delta(q,a)$ is instead a (finite) set of possibilities; equivalent because we can do BFS or IDDFS (by storing queued states separated by '#'))
- **Church-Turing thesis**: Whatever can be computed by an algorithmic devise (either function computation or language acceptance) can be done by a Turing machine
- **Countability of strings**: for each string x over $\{0,1\}^*$, let $1x$(in binary) $- 1$ be its code; let w_i be the i^{th} string
- **Countability of Turing machines**: (proof omitted); let M_i be the i^{th} machine
- **Non-RE language by diagonalisation**: $L_d := \{w_i \mid w_i \notin L(M_i)\}$ is not RE (proof: show that $\forall j \in \mathbb{N}$, $L(M_k) \neq L_d$)
- **Thm**: L is recursive $\implies \overline{L}$ is recursive
- **Thm**: L is recursive $\iff L$ is RE and $\overline{L}$ is RE
- **Universal turing machine**: $L_u := \{(M,w) \mid M \text{ accepts } w\}$ is RE
- **Thm**: $\overline{L_u}$ is not RE
- **Cor**: L_u is not recursive
- **Reduction**: P_1 reduces to P_2 ($P_1 \leq_m P2$): $\exists$ recursive f such that $x \in P_1 \iff f(x) \in P_2$ (note: we can't manipulate the answer from the oracle)
  - If P_1 is not recursive then P_2 is not recursive
  - If P_1 is not RE then P_2 is not RE
  - If P_2 is recursive then P_1 is recursive
  - If P_2 is RE then P_1 is RE
- **Machines accepting the empty language**:
  $L_e := \{M \mid L(M) = 0\}$
  $L_{ne} := \{M \mid L(M) \neq 0\}$
  Thm: L_{ne} is RE
  Thm: L_e is not recursive
  Cor: L_e is not RE
- **Non-trivial property** about RE languages: there exists at least one RE language which satisfies the property and at least one RE language which does not satisfy the property
- **Rice's thm**: If P is a non-trivial property about RE languages, then $L_P := \{M \mid L(M) \text{ satisfies property } P\}$ is undecidable
- **Post's correspondence problem (PCP)**: Given two lists of strings $A = w_1, \ldots, w_k$ and $B = x_1, \ldots, x_k$, do there exist $i_1, \ldots, i_m$ (where $m > 0$) such that $w_{i_1} \cdots w_{i_m} = x_{i_1} \cdots x_{i_m}$?
- **Modified Post's correspondence problem (MPCP)**: Given two lists of strings $A = w_1, \ldots, w_k$ and $B = x_1, \ldots, x_k$, do there exist $i_1, \ldots, i_m$ (where $m \geq 0$) such that $w_1 w_{i_1} \cdots w_{i_m} = x_1 x_{i_1} \cdots x_{i_m}$?
- **Thm**: $L_u \leq_m MPCP \leq_m PCP$
- **Thm**: $PCP \leq_m$ (Is grammar ambiguous?)
- **Further undecidable problems**:
  - Given CFGs G_1 and G_2, whether $L(G_1) \cap L(G_2) = \varnothing$?
  - Given CFGs G_1 and G_2, whether $L(G_1) = L(G_2)$?
  - Given CFG G and regular expression R, whether $L(G) = L(R)$?

Unrestricted Grammars

- $G := (N, \Sigma, S, P)$, where:
  - N is a finite set of variables (aka. non-terminals)
  - Σ is a finite set of terminals (where $N \cap \Sigma = \varnothing$)
  - P is a finite set of productions of the form $\alpha \to \beta$, where $\alpha \in (N \cup \Sigma)^* N (N \cup \Sigma)^*$ (i.e. α has at least one non-terminal) and $\beta \in (N \cup \Sigma)^*$
  - $S \in V$ is the start symbol (note: S can be implicitly start)
- **Context-sensitive grammar**: If we additionally have $|\alpha| \leq |\beta|$ for all productions $\alpha \to \beta$ in P, then G is context-sensitive
- **Thms**:
  - If G is an unrestricted grammar, then $L(G)$ is RE
  - If L is RE, then there exists an unrestricted grammar such that $L = L(G)$

Complexity

- **Time complexity**:
  - <u>$Time_M(x)$</u>: number of steps used by a machine M on input x before halting (if it does not halt, then $Time_M(x) = \infty$)
  - for non-deterministic machines, we use the maximum time on any path, including non-accepting ones
  - M is $T(n)$ time bounded, if for any input x of length n, $Time_M(x) \leq T(n)$
- **Space complexity**:
  - <u>$Space_M(x)$</u>: maximum number of cells touched by M on input x (excluding read-only input tape and one-way write-only output tape) (if it does not halt, then $Space_M(x) = \infty$)
  - M is $S(n)$ time bounded, if for any input x of length n, $Space_M(x) \leq S(n)$
- **Language classes**
  <u>$DSPACE(S(n))$</u> $:= \{L \mid \text{some } S(n) \text{ space bounded deterministic machine accepts } L\}$
  <u>$DTIME(S(n))$</u> $:= \{L \mid \text{some } T(n) \text{ time bounded deterministic machine accepts } L\}$
  <u>$NSPACE(S(n))$</u> $:= \{L \mid \text{some } S(n) \text{ space bounded nondeterministic machine accepts } L\}$
  <u>$NTIME(S(n))$</u> $:= \{L \mid \text{some } T(n) \text{ time bounded nondeterministic machine accepts } L\}$
  (for larger than n, the constant doesn't matter)
- **Arbitrarily difficult problems**: For any recursive function f, there exists a recursive function g such that no $f(n)$ time bounded machine can compute g
- **Fully space/time constructible functions**
  - $S(n)$ is <u>fully space constructible</u>: there exists a $S(n)$ space bounded TM M such that, on all inputs of length n, it uses exactly $S(n)$ space
  - $T(n)$ is <u>fully time constructible</u>: there exists a $T(n)$ time bounded TM M such that, on all inputs of length n, it uses exactly $T(n)$ time
- **Thms**
  - $DTIME(S(n)) \subseteq DSPACE(S(n))$
  - If $L \in DSPACE(S(n))$ and $S(n) \geq \log n$, then there exists $c = c(L)$ such that $L \in DTIME(c^{S(n)})$
  - If $L \in NTIME(T(n))$, then there exists $c = c(L)$ such that $L \in DTIME(c^{T(n)})$
- **Thm**: If L is accepted by a $S(n) \geq \log n$ space bounded machine, then L can be accepted by a $S(n)$ space bounded machine which halts on all inputs
- **Space hierarchy theorem**: If $S_2(n), S_1(n) \geq \log n$ and $S_2(n)$ is fully space constructible and $\lim_{n\to\infty} \frac{S_1(n)}{S_2(n)} = 0$, then $DSPACE(S_2(n)) - DSPACE(S_1(n)) \neq \varnothing$
- **Time hierarchy theorem**: If $T_2(n), T_1(n) \geq (1+\varepsilon)n$ and $T_2(n)$ is fully time constructible and $\lim_{n\to\infty} \frac{T_1(n)\log(T_1(n))}{T_2(n)} = 0$, then $DTIME(T_2(n)) - DTIME(T_1(n)) \neq \varnothing$

NP-completeness

- $\mathbf{P} := \{L \mid \text{some polynomial time bounded deterministic machine accepts } L\}$
- $\mathbf{NP} := \{L \mid \text{some polynomial time bounded nondeterministic machine accepts } L\}$
- $\mathbf{coNP} := \{L \mid \overline{L} \in \mathbf{NP}\}$
- **"Certificate" for NP problems**: If $L \in \mathbf{NP}$, then there exists a deterministic polynomial time computable predicate $P(x,y)$ and a polynomial q such that $x \in L \iff (\exists y \mid |y| \leq q(|x|))\,[P(x,y)]$
- **Polynomial-time many-to-one reducibility**: $L_1 \leq_m^p L_2$: $\exists$ polynomial time computable f such that $x \in L_1 \iff f(x) \in L_2$
- L is **NP-hard**: $\forall L' \in \mathbf{NP}, L' \leq_m^p L$
- L is **NP-complete**: $L \in \mathbf{NP}$ and L is **NP**-hard