Automata & Computability Final Notes

$\emptyset$ = empty set

ε = null string

Strings & Sets:

→ Decision Problem: Function with a one-bit output where the input is a string. → one must specify set A of possible inputs & $B \subseteq A$

→ Always take the set of Possible Inputs = Set of finite-length strings over some fixed finite alphabet

→ Alphabet: Any finite set of characters Σ

→ String over Σ: Any finite-length sequence of elements of Σ — Ex: if $\Sigma = \{a, b\}$, $aabab \in \Sigma^*$

↳ length of string $x \in \Sigma^*$ is the number of symbols, $|x|$

↳ ε is empty string of length 0

↳ Let $a \in \Sigma$, a^n is a string of a's of length n

→ String Concatenation: $x, y \in \Sigma^* \rightarrow xy$ is the concatenation

↳ Associativity: $x(yz) = (xy)z$

↳ ε is identity for concat. $\varepsilon x = x\varepsilon = x$

↳ $|xy| = |x| + |y|$

↳ Special Case: $\forall m, n \in \mathbb{N} \cdot a^m a^n = a^{m+n}$

→ Prefix of string $x \in \Sigma^*$ = Initial substring of x, y, s.t. $\exists z \, (x = yz)$

→ Proper Prefix of String $x \in \Sigma^*$ = $\forall x \in \Sigma^*$ · a proper prefix y is a prefix s.t. $y \neq \varepsilon \wedge y \neq x$)

→ Cardinality of Sets: $\forall A$ set · $|A|$ = # of elements $|\emptyset| = 0$

→ Set Union: $\forall A, B \, (A \cup B \triangleq \{x \mid x \in A \vee x \in B\})$ → Set Complement in Σ^*: $\forall A \subseteq \Sigma^* \, (\sim A = \{x \in \Sigma^* \mid x \notin A\})$

→ Set Intersection: $\forall A, B \, (A \cap B \triangleq \{x \mid x \in A \wedge x \in B\})$ → Set Concatenation: $\forall A, B \, (AB = \{xy \mid x \in A \wedge y \in B\})$

→ Power of Set: For set A, A^n is defined inductively as follows: Ex: $\{a, ab\}\{b, ba\} = \{ab, aba, abb, abba\}$

$A^0 = \{\varepsilon\}$ $A^n = \overbrace{AA \cdots A}$ $A^{n+1} = AA^n$ $A^{m+n} = A^m A^n$

→ Asterate A^* of set A: Union of all finite nonnegative powers of A $\forall A \, (A^* = \bigcup_{n \geq 0} A^n = A^0 \cup A^1 \cup A^2 \cup \cdots)$

↳ $A^* = \{x_1 x_2 \cdots x_n \mid n \geq 0 \text{ and } x_i \in A, 1 \leq i \leq n\}$

→ A^+ of set A: Union of all finite nonzero powers of A $\forall A \, (A^+ = AA^* = \bigcup_{n \geq 1} A^n)$

→ Set Union, Intersection, Concat are associative → $\emptyset$ is an identity of $\cup$: $\emptyset \cup A = A \cup \emptyset = A$

$A \cup (B \cup C) = (A \cup B) \cup C$ → The set $\{\varepsilon\}$ is an identity of concat: $\{\varepsilon\}A = A\{\varepsilon\} = A$

$A \cap (B \cap C) = (A \cap B) \cap C$ → Set Union & Intersection are commutative:

$A(BC) = (AB)C$ $A \cup B = B \cup A$ $A \cap B = B \cap A$

→ $\emptyset$ is an Annihilator for concat: $\emptyset A = A\emptyset = \emptyset$ → ONLY Set concat distributes over Union

→ Set Union & Intersection distribute over each other

$A \left(\bigcup_{i \in I} B_i \right) = \bigcup_{i \in I} AB_i$

$\left(\bigcup_{i \in I} B_i \right) A = \bigcup_{i \in I} B_i A$

*: Properties:

$A^* \triangleq \{\varepsilon\} \cup AA^*$

$A^*A^* \triangleq A^*$

$A^{**} \triangleq A^*$

$\emptyset^* \triangleq \{\varepsilon\}$

Finite Automata & Regular Sets:

→ State of a System: Instantaneous Description of a system (gives all relevant info to determine how a system can evolve from that state)

→ Transitions: Change of state (spontaneous or response to inputs) → assumed to be instantaneous.

→ Finite-State Machine: System that consists only of finitely many states & transitions → modelled via finite automatons.

→ Deterministic Finite Automaton (DFA): Structure $M = (Q, \Sigma, \delta, s, F)$ $F \subseteq Q$ (accept states)

- set of states Q
- start state $s \in Q$
- finite alphabet Σ
- $\delta : Q \times \Sigma \to Q$ ← Transition f^n that tells us which state to move to in response of input

→ How a finite automaton works: Input $x \in \Sigma^*$ Start at s. Scan input $x \in \Sigma^*$ from left to right, one at a time, moving a state pointer according to δ. At the end of the string, the state pointer is at some state p.

↳ $x \in \Sigma^*$ is accepted if $p \in F$ ↳ $x \in \Sigma^*$ is rejected if $p \notin F$

→ Define Transition Function δ inductively: $\hat{\delta} : Q \times \Sigma^* \to Q$ from δ by induction on $|x|$ where $x \in \Sigma^*$

$\hat{\delta}(q, x)$ is state M that ends up when started at q and fed $x \in \Sigma^*$.

$\hat{\delta}(q, \varepsilon) = q$

$\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$

↳ Base Case: Machine moves nowhere at ε input.

↳ Inductive Step: State reachable from q under xa is state reachable from p under a, p is state reachable from q under x

→ $x \in \Sigma^*$ is accepted if $\hat{\delta}(s, x) \in F$ and rejected if $\hat{\delta}(s, x) \notin F$

→ Language: $L(M)$ = set or language accepted by M = set of all strings accepted by M = $\{x \in \Sigma^* \mid \hat{\delta}(s, x) \in F\}$

→ $A \subseteq \Sigma^*$ is a Regular Set if $A = L(M)$ for some M : DFA.

→ We must show that if A & B are regular, then $A \cap B$, $A \cup B$, AB and $\sim A$ are regular

The Product Construction:

→ Assume A and B are regular sets. Then there are automata $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ s.t. $L(M_1) = A$, $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ s.t. $L(M_2) = A$

To show $A \cap B$ is regular, we must build M_3 s.t. $L(M_3) = A \cap B$.

Let $M_3 = (Q_3, \Sigma, \delta_3, s_3, F_3)$ where $Q_3 = Q_1 \times Q_2$, $F_3 = F_1 \times F_2$, $s_3 = (s_1, s_2)$

$\delta_3 : Q_3 \times \Sigma \to Q_3$ $\delta_3((p, q), \varepsilon) = (p, q)$

↳ s.t. $\delta_3((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$

→ Lemma (A): $\forall x \in \Sigma^* . \ \hat{\delta}_3((p,q),x) = (\hat{\delta}_1(p,x), \hat{\delta}_2(q,x))$

Proof: By induction on $|x| \in \mathbb{N}$:

Base Case: For $x = \varepsilon$, $\hat{\delta}_3((p,q),\varepsilon) = (p,q) = (\hat{\delta}_1(p,\varepsilon), \hat{\delta}_2(q,\varepsilon))$

Inductive Step: Assume lemma holds for $x \in \Sigma^*$. WTS it holds for $xa \in \Sigma^*$ where $a \in \Sigma$:

$$\begin{aligned}
\hat{\delta}_3((p,q),xa) &= \delta_3(\hat{\delta}_3((p,q),x), a) \\
&= \delta_3((\hat{\delta}_1(p,x), \hat{\delta}_2(q,x)), a) \\
&= (\delta_1((\hat{\delta}_1(p,x), \hat{\delta}_2(q,x)), a), \ \delta_2((\hat{\delta}_1(p,x), \hat{\delta}_2(q,x)), a) \\
&= (\hat{\delta}_1(p,xa), \hat{\delta}_2(q,xa))
\end{aligned}$$

→ Theorem: $L(M_3) \triangleq L(M_1) \cap L(M_2)$

Proof: $\forall x \in \Sigma^*$:

$$\begin{aligned}
& x \in L(M_3) \\
\equiv \ & \hat{\delta}_3(s_3, x) \in F \\
\equiv \ & \hat{\delta}_3((s_1,s_2),x) \in F \\
\equiv \ & (\hat{\delta}_1(s_1,x), \hat{\delta}_2(s_2,x)) \in F \\
\equiv \ & \hat{\delta}_1(s_1,x) \in F \wedge \hat{\delta}_2(s_2,x) \\
\equiv \ & x \in L(M_1) \wedge x \in L(M_2) \\
\equiv \ & x \in (L(M_1) \cap L(M_2))
\end{aligned}$$

To show that regular sets are closed under $\sim A$, take a DFA accepting A and interchange set of Accept & Non-Accept states.

$M = (Q, \Sigma, \delta, s, F)$ s.t. $A = L(M)$

$M' = (Q, \Sigma, \delta, s, Q \setminus F)$ s.t. $\sim A = L(M')$

Non-Deterministic Finite Automata

→ Nondeterminism: Situations in which the next state of computation is not uniquely determined by a current state (arises in real life when there exists incomplete info bout a state that can affect the course of computation.)

↳ important in the design of efficient algorithms: ↳ May not know how a computation evolves, but we have some idea of range of possibilities

→ Fact: In terms of sets accepted, NFA is no more powerful than DFAs $\equiv \forall N: \text{NFA}, \exists D: \text{DFA} \cdot L(D) = L(N)$

→ Nondeterministic Finite Automata: Next state is not necessarily uniquely determined by current state input

↳ accepts $x \in \Sigma^*$ if at least 1 computation path on x from at least 1 start state leads to an accept state (rejects otherwise)

→ To show an NFA accepts set B, we must argue that $\forall x \in B$ (there is a path of transitions that goes from start → accept when end of x reached)

→ Formal Def^n of NFA = Five Tuple $N \triangleq (Q, \Sigma, \Delta, S, F)$ — Q: set of states; Σ: finite alphabet; Δ: set of start states S; F: set of accept states

→ Δ extended to $\hat{\Delta}$:

$\Delta: Q \times \Sigma \to 2^Q$ where $2^Q \triangleq \{A \mid A \subseteq Q\}$

$\hat{\Delta}: 2^Q \times \Sigma^* \to 2^Q$

$\hat{\Delta}(A, \varepsilon) \triangleq A$

$\forall A \subseteq Q \ \forall x \in \Sigma^* \ \forall a \in \Sigma \ (\hat{\Delta}(A, xa) \triangleq \bigcup_{q \in \hat{\Delta}(A,x)} \Delta(q,a))$ — set of all states reachable under x from some state in A

$\forall p \in Q \ \forall a \in \Sigma \ (\Delta(p,a) =$ set of states N is allowed to move to under a$)$

$q \in \hat{\Delta}(A,x)$ if N can move from $p \in A$ to q under x

$\hat{\Delta}(A,a) = \bigcup_{p \in A} \Delta(p,a)$

→ NFA N is said to accept $x \in \Sigma^*$ if $\hat{\Delta}(s,x) \cap F \neq \emptyset$

≡ N accepts x if there exists $q \in F$ s.t. q reachable from $s \in S$

$L(N) \triangleq \{ x \in \Sigma^* \mid N \text{ accepts } x \} \triangleq \{ x \in \Sigma^* \mid \hat{\Delta}(s,x) \cap F \neq \emptyset \}$

↳ $\forall M : \text{DFA} \mid M = (Q, \Sigma, \delta, s, F) \cdot \left(\exists N : \text{NFA} \mid N = (Q, \Sigma, \Delta, \{s\}, F) \cdot M \equiv N \text{ where } \Delta(p,a) \triangleq \{ \delta(p,a) \} \right)$

→ Lemma 6.1: $\forall x,y \in \Sigma^* \; \forall A \subseteq Q \; \left(\hat{\Delta}(A, xy) = \hat{\Delta}(\hat{\Delta}(A,x), y) \right)$

Proof by induction on $|y| \in \mathbb{N}$:

Basis: For $y = \epsilon$, $\hat{\Delta}(A, x\epsilon) = \hat{\Delta}(A,x) = \hat{\Delta}(\hat{\Delta}(A,x), \epsilon)$

Induction Step: Assume $\forall x,y \in \Sigma^* \; \forall A \subseteq Q \; \left(\hat{\Delta}(A,xy) = \hat{\Delta}(\hat{\Delta}(A,x), y) \right)$

$\forall y \in \Sigma^* \; \forall a \in \Sigma$:

$$\hat{\Delta}(A, xya) = \bigcup_{q \in \hat{\Delta}(A,xy)} \Delta(q,a)$$

$$= \bigcup_{q \in \hat{\Delta}(\hat{\Delta}(A,x),y)} \Delta(q,a)$$

$$= \hat{\Delta}\left(\hat{\Delta}(A,x), ya\right)$$

∴ Hence proven.

→ Lemma 6.2: $\hat{\Delta}$ commutes with set union. $\forall A_i \subseteq Q, x \in \Sigma^* \left(\hat{\Delta}(\bigcup_i A_i, x) = \bigcup_i \hat{\Delta}(A_i, x) \right)$

↳ Proof by induction on $|x| \in \mathbb{N}$:

Basis: For $x = \epsilon$, $\hat{\Delta}(\bigcup_i A_i, \epsilon) = \bigcup_i A_i = \bigcup_i \hat{\Delta}(A_i, \epsilon)$

Induction Step: Assume $\forall A_i \subseteq Q, x \in \Sigma^* \left(\hat{\Delta}(\bigcup_i A_i, x) = \bigcup_i \hat{\Delta}(A_i, x) \right)$

$\forall x \in \Sigma^* \; \forall A_i \subseteq Q \; \forall a \in \Sigma$:

$$\hat{\Delta}(\bigcup_i A_i, xa)$$

$$= \bigcup_{q \in \hat{\Delta}(\bigcup_i A_i, x)} \Delta(q,a)$$

$$= \bigcup_{q \in \bigcup_i \hat{\Delta}(A_i, x)} \Delta(q,a)$$

$$= \bigcup_i \bigcup_{q \in \hat{\Delta}(A_i, x)} \Delta(q,a)$$

$$= \bigcup_i \hat{\Delta}(A_i, xa)$$

Hence proven.

→ Theorem: Set A is regular ≡ $\exists N : \text{NFA} \cdot L(N) = A$

The Subset Construction Game:

→ Theorem: $\forall N : \text{NFA} \; \forall A \text{ set} \left(A = L(N) \Rightarrow \exists M : \text{DFA} \left(L(M) = L(N) = A \right) \right)$ — A is regular.

→ Assume $N = (Q_N, \Sigma, \Delta_N, S_N, F_N)$ and $M = (Q_M, \Sigma, \delta_M, s_M, F_M)$

then $Q_M \triangleq 2^{Q_N} \triangleq \{ B \subseteq Q_N \}$

$s_M \triangleq S_N$

$F_M \triangleq \{ A \subseteq Q_N \mid A \cap F_N \neq \emptyset \}$

$\forall A \subseteq Q_N \; \forall x \in \Sigma^* \left(\delta_M(A,x) = \hat{\Delta}(A,x) \right)$

→ Lemma 6.3: $\forall A \subseteq Q_N \; \forall x \in \Sigma^* \; (\hat{\delta}_M(A,x) = \hat{\Delta}_N(A,x))$

Proof by induction on $|x| \in \mathbb{N}$:

Basis: For $x = \varepsilon$, $\hat{\delta}_N(A,\varepsilon) = A = \hat{\Delta}_N(A,\varepsilon)$

Induction Step: Assume $\forall A \subseteq Q_N \; \forall x \in \Sigma^* \; (\hat{\delta}_M(A,x) = \hat{\Delta}_N(A,x))$

$\forall A \subseteq Q_N \forall x \in \Sigma^* \forall a \in \Sigma$:

$$\hat{\delta}_M(A, xa) = \delta_M(\hat{\delta}_M(A,x), a)$$
$$= \delta_M(\hat{\Delta}_N(A,x), a)$$
$$= \hat{\Delta}_N(\hat{\Delta}_N(A,x), a)$$
$$= \hat{\Delta}_N(A, xa) \text{ by 6.1}$$

→ Theorem 6.4: The automaton M and N accept the same set.

Proof: $\forall x \in \Sigma^* : x \in L(M)$

$$\equiv \hat{\delta}_M(s_M, x) \in F_M$$
$$\equiv \hat{\Delta}_N(s_N, x) \cap F_N \neq \emptyset \text{ by 6.3, Def}^{\underline{n}} \text{ of } s_N, \text{ Def}^{\underline{n}} \text{ of } F_M$$
$$\equiv x \in L(N)$$

→ ε-Transition: Transition $p \xrightarrow{\varepsilon} q$ (only for NFAs). Automaton can take such a transition w/o reading input

→ We extend $\Delta : Q \times \Sigma \to 2^Q$ to $\Delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$)

→ Theorem: Any NFA with ε-transitions can be converted into an equivalent NFA (or DFA via subset construction)

Pattern Matching:

→ Pattern: String of symbols of a certain form representing finite or infinite set of strings in Σ^* (set of patterns defined inductively)

↳ Atomic or Compound patterns.

→ We tell which strings $x \in \Sigma^*$ match patterns. $L(\alpha) = \{x \in \Sigma^* \mid x \text{ matches } \alpha\}$

Binary Operators: $+, \cdot, \cap$

Unary Operators: $\sim, ^*, ^+$

→ Atomic patterns are the following:

↳ a for each $a \in \Sigma$ $\quad x \in L(a) \equiv x \in \{a\}$

↳ ε $\quad x \in L(\varepsilon) \equiv x \in \{\varepsilon\}$

↳ $\emptyset$ $\quad x \in L(\emptyset) \equiv x \in \emptyset$

↳ #, $\quad x \in L(\#) \equiv x \in \Sigma$

↳ @, $\quad x \in L(@) \equiv x \in \Sigma^*$

→ Compound Patterns formed inductively using

↳ $x \in L(\alpha + \beta) \equiv x \in (L(\alpha) \cup L(\beta))$

↳ $x \in L(\alpha \cap \beta) \equiv x \in (L(\alpha) \cap L(\beta))$

↳ $x \in L(\alpha\beta) \equiv x \in [L(\alpha) L(\beta)) \equiv x \in \{yz \mid y \in L(\alpha) \wedge z \in L(\beta)\}$

↳ $x \in L(\sim\alpha) \equiv x \in (\sim L(\alpha)) \equiv x \in (\Sigma^* - L(\alpha))$

↳ $x \in L(\alpha^*) \equiv x \in (L(\alpha))^* \equiv x \in \{x_1 x_2 \cdots x_n \mid n \in \mathbb{N} \wedge x_i \in L(\alpha), 1 \leq i \leq n\}$

↳ $x \in L(\alpha^+) \equiv x \in (L(\alpha))^+ \equiv x \in \{x_1 x_2 \cdots x_n \mid n \in \mathbb{Z} \wedge x_i \in L(\alpha), 1 \leq i \leq n\}$

Examples: $\Sigma^* = L(@) = L(\#^*)$

Singleton sets: if $x \in \Sigma^*$, $L(x) = \{x\}$ → x is a pattern

Finite sets: if $x_1, x_2 \ldots, x_m \in \Sigma^*$, $L(x_1 + x_2 + \cdots + x_m) = \{x_1, x_2, \ldots, x_m\}$

Strings containing AT LEAST 3 occurences of a: $@a@a@a@$

All single letters EXCEPT a: $\# \cap \sim a$

Strings WITH NO OCCURENCE OF a: $(\# \cap \sim a)^*$

$L(\beta) = L(\alpha)$ but that doesn't mean $\alpha = \beta$

↙ How to tell if 2 patterns are equivalent?

→ Patterns are redundant → We actually do not need all of the atomic or compound patterns

Ex: If $\Sigma = \{a,b,c\}$ we can replace $\#$ with $a+b+c$, ϵ with $\emptyset^*$, $\alpha \cap \beta$ with $\neg(\neg\alpha + \neg\beta)$

$\epsilon \equiv \neg(\#@) \equiv \emptyset^*$ $\quad \alpha^+ \equiv \alpha\alpha^*$

$@ \equiv \#^*$

Ex: $ac^*b + a^*b^* \equiv (a(c^*)b) + ((a)^*(b)^*)$

Regular Expressions:

→ Regular Expression = Pattern that only uses the following:

- Atomic: $\emptyset$, a for each $a \in \Sigma$, ϵ
- Operator: $+$, $\cdot$, * — order of precedence: $^* > \cdot > +$

→ Regular Expressions define equivalency classes: $\alpha \equiv \beta \equiv L(\alpha) = L(\beta)$

↳ Reflexivity: $\alpha \equiv \alpha$ ↳ Symmetry: $(\alpha \equiv \beta) \equiv (\beta \equiv \alpha)$ ↳ Transitivity: $\alpha \equiv \beta \wedge \beta \equiv r \Rightarrow \alpha \equiv r$

↳ Partial Order: $\alpha \leq \beta$ if $L(\alpha) \subseteq L(\beta)$

→ Theorem: Let $A \subseteq \Sigma^*$. Then the following statements are equivalent

i) A is regular (has a DFA)

ii) $\exists N : \text{NFA} \cdot L(N) = A$

iii) $\exists N : \text{NFA} \mid N = (Q, \Sigma, \Delta, S, F)$ s.t. $\Delta : Q \times (\Sigma \cup \{\epsilon\}) \to Q \cdot L(N) = A$

iv) $\exists \alpha : \text{pattern} \cdot L(\alpha) = A$

v) $\exists \gamma : \text{regex} \cdot L(\gamma) = A$

Proof:

i) ⇒ ii): $\forall M : \text{DFA}, \exists N : \text{NFA} \ (L(M) = L(N) = A)$

ii) ⇒ i): $\forall N : \text{NFA} \ \forall A \ (A = L(N) \Rightarrow \exists M : \text{DFA} \ (L(M) = L(N) = A))$ — subset construction

ii) ⇒ iii): Trivial. Any NFA N has a corresponding NFA N' with ϵ-transitions.

iii) ⇒ ii): Reverse.

v) ⇒ iv): Regular Expressions are patterns. ∴ if there exist regex α s.t. $L(\alpha) = A$ then there exists a pattern.

iv) ⇒ v): Possible to recreate patterns as regular expressions.

v) ⇒ iii): If regex γ is atomic then finding an NFA is trivial.

$\alpha = \emptyset \Rightarrow$ 

$\alpha = a \Rightarrow$ 

$\alpha = \epsilon \Rightarrow$

Otherwise we use structural induction: Assume $L(\alpha)$ and $L(\beta)$ are regular (we have NFAs for them).

Then
$$\begin{cases} L(\alpha\beta) = L(\alpha)L(\beta) \\ L(\alpha+\beta) = L(\alpha) \cup L(\beta) \\ L(\alpha^*) = L(\alpha)^* \end{cases}$$

Combine the two NFAs depending on the operators used.

Laws to simplify regular expressions:

$\alpha + (\beta + \gamma) \equiv (\alpha + \beta) + \gamma$

$\alpha + \beta \equiv \beta + \alpha$

$\alpha + \phi \equiv \alpha$

$\alpha + \alpha \equiv \alpha$

$\alpha(\beta\gamma) \equiv (\alpha\beta)\gamma$

$\varepsilon\alpha \equiv \alpha\varepsilon \equiv \alpha$

$\alpha(\beta + \gamma) \equiv \alpha\beta + \alpha\gamma$

$(\alpha + \beta)\gamma \equiv \alpha\gamma + \beta\gamma$

$\phi\alpha \equiv \alpha\phi \equiv \phi$

$\varepsilon + \alpha\alpha^* \equiv \alpha^*$

Partial Orders:

$\beta + \alpha\gamma \leq \gamma \Rightarrow \alpha^*\beta \leq \gamma$

$\beta + \gamma\alpha \leq \gamma \Rightarrow \beta\alpha^* \leq \gamma$

Other Equations:

$(\alpha\beta)^*\alpha \equiv \alpha(\beta\alpha)^*$

$(\alpha^*\beta)^*\alpha^* \equiv (\alpha + \beta)^*$

$\alpha^*(\beta\alpha^*)^* \equiv (\alpha + \beta)^*$

$(\alpha + \varepsilon)^* = \alpha^*$

$\alpha\alpha^* \equiv \alpha^*\alpha$

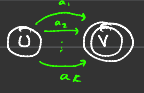
Converting Automata to Regular Expressions:

→ Let N: NFA $= (Q, \Sigma, \Delta, S, F)$.

↳ Idea: $\forall u, v \in Q \; \forall R \subseteq Q \; (A^R_{u,v} \triangleq$ set of strings that take N from u to v by only using states in R)

$\alpha^R_{u,v}$: a regular expression s.t. $L(\alpha^R_{u,v}) = A^R_{u,v}$

Base Case: For $R = \phi$



$\therefore \alpha^{\phi}_{u,v} = \begin{cases} \varepsilon & u = v, k = 0 \\ \phi & u \neq v, k = 0 \\ \varepsilon + a_1 + \cdots + a_k & u = v, k > 0 \\ a_1 + \cdots + a_k & u \neq v, k > 0 \end{cases}$

$\rightarrow \forall R \subseteq Q \mid R \neq \phi \cdot \forall u, v \in Q \cdot \exists q \in R \left(\alpha^R_{u,v} = \alpha^{R-\{q\}}_{u,v} + \alpha^{R-\{q\}}_{u,q} \left(\alpha^{R-\{q\}}_{q,q}\right)^* \alpha^{R-\{q\}}_{q,v} \right)$

→ To convert any NFA $N = (Q, \Sigma, \Delta, S, F)$ into a regular expression

Let $s \in S$ and $f \in F$. Then $\alpha^Q_{s,f} = \alpha^{Q-\{q\}}_{s,f} + \alpha^{R-\{q\}}_{s,q} \left(\alpha^{R-\{q\}}_{q,q}\right)^* \alpha^{R-\{q\}}_{q,f}$

When doing practice, choose q state of the highest degree.

Limitations of Finite Automata:

→ Classic example of a nonregular set: $B = \{a^n b^n \mid n \geq 0\} = \{\varepsilon, ab, aabb, \cdots\}$

impossible to do for infinitely long strings with finite memory.

↳ For an automaton M to accept B, it would have to remember how many a's it has seen before checking the number of b's

→ Proof of why B is nonregular: By contradiction.

Assume B is regular. Then $\exists M$: DFA $(L(M) = B)$. Let there be k states in M. Consider $a^n b^n$ input where $n >> k$

$a^n b^n = \overbrace{aaa \cdots aaa}^{n} \overbrace{bbb \cdots bbb}^{n}$

Since $n >> k$, by the pigeonhole principle, there must be a state p s.t. it is repeated while scanning the input. $\therefore v \neq \varepsilon$

Break up $a^n b^n$ into 3 pieces: u, v, w where v is the string of a's scanned between 2 occurrences of p.

$\overbrace{aa \cdots aa}^{u} \overbrace{aa \cdots aa}^{v} \overbrace{aaa \cdots aaa\, bbb \cdots bbb}^{w}$

$p \quad p$

$\hat{\delta}(s, u) = p \quad \hat{\delta}(p, v) = p \quad \hat{\delta}(p, w) = f \in F$

v could be removed or repeated where $uv^i w$, $i \in \mathbb{N}$, $i \neq 1$

This contradicts such $uv^i w \in B$ ← and M would accept it.

since $\#a \neq \#b$

→ Proof of why $C = \{a^{2^n} \mid n \geq 0\}$ is nonregular: By contradiction, assume $\exists M$: DFA $(L(M) = C)$. Let $k = |Q_M|$ and $n >> k$ for an input a^{2^n} into M. Since $n >> k$, by the pigeonhole principle, M must repeat a state p while scanning first n symbols of a^{2^n}. Thus $2^n = i + j + m$ with $0 < j \leq n$ and $\hat{\delta}(s, a^i) = p \quad \hat{\delta}(p, a^j) = p \quad \hat{\delta}(p, a^m) = f \in F$

We can add an extra a^j to get $a^{2^n + j}$ which would be accepted by M. Contradiction. $2^n < 2^n + j < 2^n + 2^n = 2^{n+1}$

$\therefore$ Since $2^n + j$ is not a power of 2, $a^{2^n + j} \notin C$.

Pomping Lemma:

→ Pomping Lemma Contrapositive Form: Let A be a set. If the following holds then set A is nonregular

$$\exists x,y,z \in \Sigma^* \left(xyz \in A \wedge |y| \geq k \wedge \forall u,v,w \in \Sigma^* \mid y = uvw \wedge v \neq \varepsilon \left(\exists i \in \mathbb{N} \mid i \neq 1 \cdot xuv^i wz \in A \right) \right)$$

if A is nonregular then for any string in A and sufficiently long substring y, y has a nonnull substring v where you can pump as many copies as you like, the resulting string is still in A.

→ Pumping Lemma as a Game: Goal is to prove A is not regular. A demon's goal is the opposite. The game is played as follows:

1. Demon picks $k \geq 0$
2. We pick $x,y,z \in \Sigma^*$ satisfying $\begin{cases} xyz \in A \\ |y| \geq k \end{cases}$
3. Demon picks $u,v,w \in \Sigma^*$ s.t. $\begin{cases} y = uvw \\ v \neq \varepsilon \end{cases}$
4. We pick i from $\{ i \in \mathbb{N} \mid i \neq 1 \}$

We win if $xuv^iwz \in A$.
We lose if $xuv^iwz \notin A$

A is not regular if we have a winning strategy no matter what demon does.

→ Trick: When trying to show set is nonregular, you can simplify problem using closure properties of regular sets.

Ex: $D = \{ x \in \{a,b\}^* \mid \#a(x) == \#b(x) \}$ For contradiction assume D is regular. $D \cap a^*b^*$ would also be regular but $D \cap L(a^*b^*) = \{ a^n b^n : n \geq 0 \}$ which is nonregular. Contradiction.

Ex: $A = \{ a^n b^m \mid n \geq m \}$ if A were regular so would be reverse(A) $= \{ b^m a^n \mid n \geq m \}$

By interchanging a's and b's, $A' = \{ a^m b^n : n \geq m \}$. However $A \cap A' = \{ a^n b^n \mid n \geq 0 \}$

Ultimate Periodicity:

→ Let $U \subseteq \mathbb{N}$. U is ultimately periodic if $\exists n \in \mathbb{N} \; \exists p \in \mathbb{Z}^+ \; \forall m \geq n \left(m \in U \equiv (m+p) \in U \right)$ (p is a period of U)

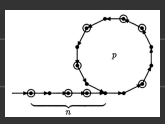
↳ In other words: except for a finite initial part (numbers $< n$), numbers are in or out of U according to a repeating pattern.

Ex: Consider $\{ 0, 3, 7, 11, 19, 20, 23, 26, 29, 32, 35, 38, 41, 44, 47, 50, \ldots \}$ → Starting at 20, every 3rd element is in set

Ultimately periodic w/ $n = 20$ $p = 3$ — n nor p are unique

→ Theorem: Let $A \subseteq \{a\}^*$. Then A is regular $\equiv \{ m \mid a^m \in A \}$, the set of lengths of strings in A, is ultimately periodic.

Proof: If A is regular then any DFA for it consists of finite tail of some length $n \in \mathbb{N}$, followed by a loop $p > 0$.



Consider any DFA for A. ∵ $\Sigma = \{a\}$ and M is DFA, there is exactly 1 edge from each state (a). ∴ There is a unique path from s to $f \in F$. Follow the path until the first time you see a repeated state. Since there are finite states, this will happen. This is a loop. $p \in \mathbb{Z}^+$ is length of loop, $n \in \mathbb{N}$ is length before loop.

$\forall a^m \mid m \geq n$ (automaton in loop after scanning a^m) $\Rightarrow$ (a^m accepted $\equiv a^{m+p}$ is accepted)

∴ If it is in same state after scanning a^m & $a^{m+p} \Rightarrow$ A is ultimately periodic.

Conversely, given up set U, let $p \in \mathbb{Z}^+$ and n be starting point of periodic behaviour. Then one can build automaton of tail length n and loop of length p accepting set of strings in $\{a\}^*$ whose lengths are in U.

→ Corollary: Let A be any regular set over Σ (doesn't have to be singleton). Then $\{ |x| \mid x \in A \}$ is ultimately periodic.

Proof: Define homomorphism $h: \Sigma \to \{a\}^*$ by $\forall b \in \Sigma \; (h(b) = a)$. Then $h(x) = a^{|x|}$. Since h preserves length, length(A) $= h(A)$

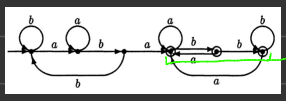
$h(A) \subseteq \{a\}^* \Rightarrow$ lengths $h(A)$ is up.

DFA State Minimization:

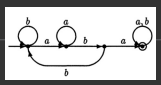
→ Some automata could be simplified either by deleting unreachable states or by collapsing states equivalent to other states.

Ex. If you apply subset construction to NFA below, you would have DFA with $2^4 = 16$ states. → However, 6 of them are unreachable

deleting unreachable states →



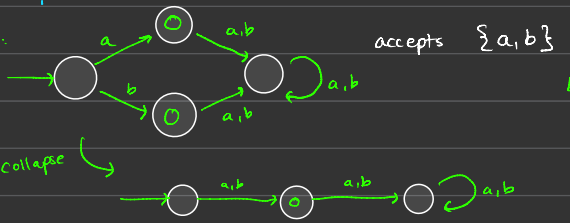
collapsed into 1 state ↓



→ Fact: every regular set has a minimal DFA unique up to isomorphism

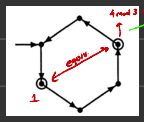
→ Say we have a DFA $M = (Q, \Sigma, \delta, s, F)$ for set A. Minimization = delete Unreachable states + collapse equiv. states.

Ex:

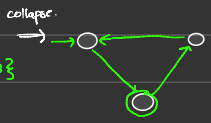


accepts $\{a, b\}$

Ex:



accepts set $\{a^m \mid m \equiv 1 \bmod 3\}$



q s.t. no $x \in \Sigma^*$ s.t. $\hat{\delta}(s, x) = q$

→ How do we know in general when 2 states can be collapsed safely

never collapse accept state p & reject state q

if $p = \hat{\delta}(s,x) \in F$ and $q = \hat{\delta}(s,y) \notin F$ then x must be accepted and y must be rejected after collapsing.

↳ if we collapse p & q, collapse $\delta(p,a)$ & $\delta(q,a)$ to maintain determinism. → ∴ do not collapse p & q.

→ Formal Proof of collapsing criterion → $\exists x \in \Sigma^* \left(\hat{\delta}(p,x) \wedge \hat{\delta}(q,x) \notin F \right)$ or vice versa

$\Rightarrow$ p & q cannot be safely collapsed.

$\approx$ is reflexive, symmetric & transitive.

Define equivalence $\approx$ on Q by: $p \approx q \triangleq \forall x \in \Sigma^* \left(\hat{\delta}(p,x) \in F \equiv \hat{\delta}(q,x) \in F \right)$

As with all equivalence relations, $\approx$ partitions set on which it is defined into disjoint equiv. classes $[p] \triangleq \{q \mid q \approx p\}$

Every element $p \in Q$ is contained in exactly 1 equiv class $[p]$ and $p \approx q \equiv [p] = [q]$

→ We now define $M/\approx$ (Quotient Automaton) = FSM whose states correspond to the equiv. class of $\approx$ → Quotient Construction.

↳ There is one state of $M/\approx$ for each $\approx$-equiv class. → Finding states in $M/\approx$ is the mathematical way to collapse states.

→ $M/\approx \triangleq (Q', \Sigma, \delta', s', F')$ where $Q' \triangleq \{[p] \mid p \in Q\}$ $\quad s' \triangleq [s]$

$\delta'([p], a) \triangleq [\delta(p,a)]$ $\quad F' \triangleq \{[p] \mid p \in F\}$

→ Lemma A: $\left(p \approx q \Rightarrow \delta(p,a) \approx \delta(q,a)\right) \equiv \left([p] = [q] \Rightarrow [\delta(p,a)] = [\delta(q,a)]\right)$

Proof: Suppose $p \approx q$. Let $a \in \Sigma$ and $y \in \Sigma^*$.

$\hat{\delta}(\delta(p,a), y) \in F$

$\equiv \hat{\delta}(p, ay) \in F$

$\equiv \hat{\delta}(q, ay) \in F$

$\equiv \hat{\delta}(\delta(q,a), y) \in F$

→ Lemma B: $pEF \equiv [p]_{\approx} \in F'$.

Proof: $pEF \Rightarrow [p]_{\approx} \in F'$ is trivial. For $[p]_{\approx} \in F' \Rightarrow pEF$ WTS $p \approx q \wedge pEF \Rightarrow q \in F$. In other words, every $\approx$-equiv. class is either subset of F or disjoint.

→ Lemma C: $\forall x \in \Sigma^* \left(\hat{\delta}'([p], x) = [\hat{\delta}(p, x)] \right)$

Proof by induction on $|x| \in \mathbb{N}$:

Basis: $x = \epsilon \Rightarrow \hat{\delta}'([p], \epsilon) = [p] = [\hat{\delta}(p, \epsilon)]$

Inductive Step: Assume $\hat{\delta}'([p], x) = [\hat{\delta}(p, x)]$ and let $a \in \Sigma$

$$\begin{aligned} \hat{\delta}'([p], xa) &= \delta'(\hat{\delta}'([p], x), a) \\ &= \delta'([\hat{\delta}(p, x)], a) \\ &= [\delta(\hat{\delta}(p, x), a)] \\ &= [\hat{\delta}(p, xa)] \end{aligned}$$

→ Theorem: $L(M/\approx) = L(M)$

Proof: For $x \in \Sigma^*$:

$$\begin{aligned} x \in L(M/\approx) &\equiv \hat{\delta}'(s', x) \in F' \\ &\equiv \hat{\delta}'([s], x) \in F' \\ &\equiv [\hat{\delta}(s, x)] \in F' \\ &\equiv \hat{\delta}(s, x) \in F \equiv x \in L(M) \end{aligned}$$

→ Doing Quotient Construction once is enough

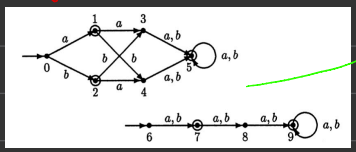
DFA State Minimization Algorithm:

→ Algorithm for collapsing relation $\approx$ for a given DFA M with no unreachable states

marked (unordered) pairs of states $\{p, q\}$ → marked as soon as reason discovered as to why p & q are not equiv.

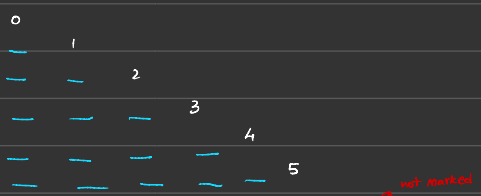
1. Write down a table of all pairs $\{p, q\}$, initially unmarked
2. Mark $\{p, q\}$ if $p \in F$ and $q \notin F$ or vice versa → if (p, q) marked here, $p \not\approx q$ are not equiv
3. Repeat following until no changes occur: if $\exists (p, q) \cdot \left((\delta(p, a), \delta(q, a)) \text{ is marked for } a \in \Sigma \right)$ then mark (p, q) — may have to consider many times until a full pass leads to no marks
4. When done, $p \approx q \equiv (p, q)$ not marked → statement to prove

Algorithm runs for finite # of steps since $\binom{n}{2}$ possible marks and make ≥ 1 new mark / pass



	a	b
→ 0	1	2
1 F	3	4
2 F	4	3
3	5	5
4	5	5
5 F	5	5

Step 1:



Step 2:

exactly one state in (p, q) is an accept state

Consider: Look at unmarked $(0,3)$ $\{0,3\} \xrightarrow{a} \{1,5\}$ not marked

∴ we don't mark $(0,3)$ yet

$\{0,3\} \xrightarrow{b} \{2,5\}$ not marked ∴ we don't mark $(0,3)$ yet.

Remaining pairs $(0,4)$, $(1,2)$ unmarked for same reasons.

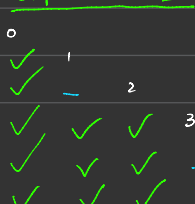
$\{1,5\} \xrightarrow{a} \{3,5\}$ is marked → mark $(1,5)$

$\{2,5\} \xrightarrow{a} \{4,5\}$ is marked → mark $(2,5)$

$\{3,4\} \xrightarrow{a,b} \{5,5\}$ never marked ∴ don't mark $(3,4)$

Step 3: Pass 1

Step 3: Pass 2:



∴ $1 \approx 2$, $3 \approx 4$

→ To check for isomorphism: Start with start state of both DFAs and call both q_0, continue labelling other states adjacent to the already labelled states & check consistency

→ **Theorem:** The pair (P, q) is marked by above algorithm

$$\equiv \exists x \in \Sigma^* \left(\hat{\delta}(p, x) \in F \text{ XOR } \hat{\delta}(q, x) \in F \right)$$

$S \subseteq Q$ where $S = \{ \{p, q\} \mid p \in F, q \notin F \}$

→ **Theorem:** Assume M_1 and M_2 are DFAs where $L(M_1) = L(M_2)$. Assume M_1' and M_2' are collapsed versions of M_1 and M_2 respectively. Then M_1' and M_2' are isomorphic.

→ Algorithm can be perceived as a finite automaton:
Let $Q = \{ \{p, q\} \mid p, q \in Q, p \neq q \}$
There are $\binom{n}{2}$ elements of Q where $n = |Q|$

$\Delta : Q \to 2^Q$ s.t. $\Delta(\{p, q\}, a) = \{ \{p', q'\} \mid p = \delta(p, a), q = \delta(q, a) \}$

Context Free Grammars and Languages:

→ **Context-Free Grammar:** Consists of a finite set of rules defining the set of well-defined expressions of some language

→ **Non-Terminal:** Generates a set of strings over Σ in a systematic or recursive way. Ex: while x ≤ y do begin x := (x+1); y := (y-1) end

→ **The language generated by a grammar** = Set of all strings not containing any nonterminals generated by the grammar → it is unambiguous if there is only derivation of a string

→ $L(G_1) \subseteq \Sigma^*$ is a **Context-Free Language**

NOTE: All regular sets are CFLs but not all CFLs are regular.

→ Formally, CFG G_1 is a 4-tuple $G_1 = (N, \Sigma, P, S)$ — Start Symbol $S \in N$; $P \subseteq N \times (N \cup \Sigma)^*$ productions; $\varepsilon \notin N$ and $\varepsilon \notin \Sigma$; $\varepsilon \in (N \cup \Sigma)^*$

We can find a CFG G_1 for every regular expression!

finite set of nonterminals; finite set of terminals

→ If $\alpha, \beta \in (N \cup \Sigma)^*$ we say β is derivable from α in one step via grammar $G_1 \equiv \alpha \xrightarrow[G_1]{1} \beta$ — If β can be obtained from α by replacing some nonterminal occurrence A in α with $\gamma \in \Sigma^*$ where $A \to \gamma \in P$

→ Let $\xrightarrow[G_1]{*}$ be the reflexive, transitive closure of relation $\xrightarrow[G_1]{}$ → $\alpha \in (N \cup \Sigma)^*$ is a sentential form if $S \xrightarrow[G_1]{*} \alpha$

→ A sentential form α is a sentence if $\alpha \in \Sigma^*$

↳ $\forall \alpha \in (N \cup \Sigma)^* \left(\alpha \xrightarrow[G_1]{0} \alpha \right)$

→ $L(G_1) = \{ x \in \Sigma^* \mid S \xrightarrow[G_1]{*} x \} \equiv L(G_1)$ is a set of all sentences of G_1

↳ $\forall \alpha, \beta \in (N \cup \Sigma)^* \exists \gamma \in (N \cup \Sigma)^* \left(\alpha \xrightarrow[G_1]{n} \gamma \wedge \gamma \xrightarrow[G_1]{1} \beta \Rightarrow \alpha \xrightarrow[G_1]{n+1} \beta \right)$

↳ $\forall \alpha, \beta \in (N \cup \Sigma)^* \exists n \in N \left(\alpha \xrightarrow[G_1]{n} \beta \Rightarrow \alpha \xrightarrow[G_1]{*} \beta \right)$

→ Ex: Palindromes $= \{ x \in \{a, b\}^* \mid x = \text{rev}(x) \}$

G_1: $S \to aSa \mid bSb \mid a \mid b \mid \varepsilon$ → $L(G_1) \subseteq B$ (induction on $n \in \mathbb{N}$ as a number of steps)

→ Ex: $A = \{ a^n b^n \mid n \geq 0 \}$ is a non-regular set but is a CFL.
$\exists G_1$: CFG s.t. $L(G_1) = A$. G_1: $S \to aSb \mid \varepsilon$

→ We can show $L(G_1) = B$ where B is a CFL by double inclusion

↳ $B \subseteq L(G_1)$ (induction on $|x|$, $x \in \mathbb{N}$ where $x \in B$)

CFGs and Normal Forms:

→ Given a string we wish to check whether it can be generated by a given CFG. ε-productions & unit productions are problematic because they cause long sentential forms consisting of more nonterminals than terminals just to derive short sentences.

→ **Def:** A CFG G_1 is in Chomsky Normal Form if all productions are of form $A \to BC$ or $A \to a$ for $a \in \Sigma$

Unit production of form $A \to B$ where $A, B \in N$

ε productions of form $A \to \varepsilon$

→ **Theorem:** $\forall G_1$: CFG $\exists G_1'$: CFG in CNF $\left(L(G_1) = L(G_1) - \{\varepsilon\} \right)$

→ To convert any CFG G_1 to CNF, we follow these steps:

1. Add productions to derive G_1' $\forall A, B \in N$ and $\alpha, \delta \in (N \cup \Sigma)^*$
   ↳ if $A \to B$ and $B \to \delta$ add $A \to \delta$ to $\hat{P}$
   ↳ if $A \to \alpha B \delta$ and $B \to \varepsilon$ add $A \to \alpha \delta$ to $\hat{P}$
2. Remove ε and unit productions
3. Remove any unreachable nonterminal productions
4. Replace terminals x with nonterminals X and add productions $X \to x$ to $\hat{P}$
5. Split long productions to fit the form $A \to BC$

Closure Properties of CFLs:

→ **Union:** If $A = L(G_1)$ and $B = L(G_2)$ for CFGs G_1 and G_2 $\Rightarrow A \cup B$ is a CFL

Ex: $S \to S_1 \mid S_2$ (Start of G_2; Start of G_1; $S_1 \to \cdots$; $S_2 \to \cdots$)

→ **Concatenation:** If $A = L(G_1)$ and $B = L(G_2)$ for CFGs G_1 and G_2 $\Rightarrow AB$ is a CFL

$S \to S_1 S_2 \cdots$

→ **Complement:** If $A = L(G_1)$, $\forall A$ is not necessarily a CFL. Sometimes complements of non-CFLs are CFLs

→ **Intersection:** If $A = L(G_1)$ and $B = L(G_2)$ for CFGs G_1, G_2 $\Rightarrow A \cap B$ is not necessarily a CFL

→ if A is regular and B is CFL $A \cap B$ is a CFL

Pumping Lemma for CFLs:

Key insight: $\forall G_1$ is in CNF (any parse tree for a very long string must have very long path) ⇒ repeated state

→ Parse Tree of $x \in \Sigma^*$: Tree representing the productions applied in derivation of x from S

→ **Pumping Lemma for CFLs:** $\forall A$ CFL $\exists k \in N \; \forall z \in A, |z| \geq k \; \exists u, v, w, x, y \in \Sigma^*$, $z = uvwxy \wedge vx \neq \varepsilon \wedge |vwx| \leq k \; \forall i \in N \left(uv^i w x^i y \in A \right)$

→ **Pumping Lemma Proof:** Let G_1 be a grammar for A in CNF. Take $k = 2^{n+1}$ where $n = |N|$ of $G_1 = (N, \Sigma, P, S)$. Suppose that for sake of contradiction that $z \in A$ and $|z| \geq k$.

Any parse tree in G_1 for z must have a depth of $\geq n+1$. Consider the longest path which has a path of $\geq n+1$, meaning $\geq n+1$ occurrences of nonterminals. By the pigeonhole principle, some nonterminals occur for more than once. Take the first pair of repeated nonterminals along the path from the bottom. Say X is the repeated nonterminal.

Break z up into $u, v, w, x, y \in \Sigma^*$ s.t. w is generated by the lower occurrence of X and vwx generated by the upper occurrence. Let T be the subtree rooted at the upper occurrence of X and let t be the subtree rooted at the lower occurrence. By replacing t with T we get a valid parse tree of uv^2wx^2y → repeat for uv^iwx^iy for $i \in N$

→ **Pumping Lemma (Contrapositive Form) for CFLs:** $\forall \text{CFL } A \; \forall k \in N \; \exists z \in A, |z| \geq k \; \forall u, v, w, x, y \in \Sigma^*, z = uvwxy \wedge |vwx| \leq k \wedge vx \neq \varepsilon \; \exists i \in N \left(uv^i w x^i y \notin A \right)$ (in different)

Parsing and the CKY Algorithm:

→ Given a CFL A and $x \in \Sigma^*$, how can we efficiently determine if $x \in A$ → Convert A to CNF. Try all derivations of $2|x|-1$ to see any of them generate x

→ Cubic Time algorithm (CKY) uses dynamic programming to determine $\forall y$ substring of x (the set of nonterminals that generate x) → Use Dynamic Programming to divide problem into smaller problems and solve the smaller problems first and use these answers to solve bigger problems. Avoid solving same sub-problem multiple steps

```
for i := 0 to n - 1 do                    /* strings of length 1 first */
  begin
    T_{i,i+1} := ∅;                       /* initialize to ∅ */
    for A → a a production of G do
      if a = z_{i+1} then T_{i,i+1} := T_{i,i+1} ∪ {A}
  end;
for m := 2 to n do                        /* for each length m ≥ 2 */
  for i := 0 to n - m do                  /* for each substring */
    begin                                 /* of length m */
      T_{i,i+m} := ∅;                     /* initialize to ∅ */
      for j := i + 1 to i + m - 1 do      /* for all ways to break */
        for A → BC a production of G do   /* up the string */
          if B ∈ T_{i,j} ∧ C ∈ T_{j,i+m}
            then T_{i,i+m} := T_{i,i+m} ∪ {A}
  end;
```

Ex: Set of all strings with #a = #b

x = aabbab (positions 0 1 2 3 4 5 6)

S → AB | BA | ss | AC | BD

A → a

B → b

C → SB

D → SA

If $T_{1,6} = S$ then $x \in A$

→ **Deterministic CFL:** Language accepted by a DPDA → transition relation is single-valued & ⊣ at end of string to tell if end of string reached

Every DCFL is a CFL but not every CFL is not a DCFL

Not closure under ∪

Nondeterministic Pushdown Automata

→ Nondeterministic Pushdown Automata: NFA with a stack (which can be used to record potentially unbounded amount of information)

↳ Input head is a read-only & moves from left to right (Machine stores info like a Stack) → In each step, top symbol popped off stack based on the top symbol, currently read symbol & current state

→ Formally, $N = (Q, \Sigma, \Gamma, \delta, s, \perp, F)$ — finite set of states; finite set of input symbols; finite set of stack symbols; start stack symbol $\perp \in \Gamma$; start state $s \in Q$; finite set of accept states $F \subseteq Q$

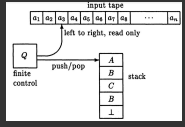
$\delta \subseteq (Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma) \times (Q \times \Gamma^*)$

push symbols onto the stack, move the head to the right & enter a new state

$((p, a, A), (q, B_1 B_2 \ldots B_k)) \in \delta$ + ε-transitions

$((p, a, A), (q, B_1 B_2 \ldots B_k)) \in \delta$ for $a \in (Q \cup \{\varepsilon\})$

≡ whenever machine is in state p after reading a and A, pop A off, push $B_1 B_2 \ldots B_k$ and enter state q



→ Configuration of NPDA M = Element of $Q \times \Sigma^* \times \Gamma^*$ describing the current state, portion of input unread, and stack contents

$((q, a, A), (q, \gamma)) \in \delta \Rightarrow \forall y \in \Sigma^* \forall \beta \in \Gamma^* \left((p, ay, A\beta) \xrightarrow{1}_M (q, y, \gamma\beta) \right)$

$((p, \varepsilon, A), (q, \gamma)) \in \delta \Rightarrow \forall y \in \Sigma^* \forall \beta \in \Gamma^* \left((p, y, A\beta) \xrightarrow{1}_M (q, y, \gamma\beta) \right)$

→ Reflexive, Transitive Relation $\xrightarrow{*}_M$:

↳ $C \xrightarrow{0}_M D \equiv C = D$

↳ $C \xrightarrow{n+1}_M D \equiv \exists E \left(C \xrightarrow{n}_M E \wedge E \xrightarrow{1}_M D \right)$

↳ $C \xrightarrow{n}_M D \equiv \exists n \in \mathbb{N} \left(C \xrightarrow{n}_M D \right)$

→ Inground. set of configs. is Ω

→ Start Configuration is always $(s, x, \perp)$ for $x \in \Sigma^*$

→ The next config relation $\xrightarrow{1}_M$ describes how machine can move from 1 config. to the next in 1 step

→ 2 Diff. Def^n of Acceptance:

↳ M accepts $x \in \Sigma^*$ by final state if $(s, x, \perp) \xrightarrow{*}_M (f, \varepsilon, \gamma)$ for some $q \in F$ and $\gamma \in \Gamma^*$

↳ M accepts $x \in \Sigma^*$ by empty stack if $(s, x, \perp) \xrightarrow{*}_M (q, \varepsilon, \varepsilon)$ for some $q \in Q$

→ If $A = L(G)$ for some CFG G, we can always find an NPDA M such that $L(M) = L(G)$ → we don't need many states. The stack contents of Configurations of NPDA M Derivations of sentential Forms as of CFG G.

To prove that this machine is correct, one must argue that for every string x, there is a sequence of transitions leading to an accept configuration from the start configuration on input x; and for every string x, no sequence of transitions leads to an accept configuration from the start configuration on input x. □

Ex: Given grammar $G = (N, \Sigma, P, S)$ s.t. P is the following productions:

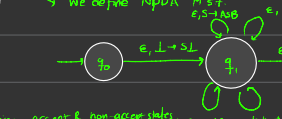
$S \to ASB \mid AB$

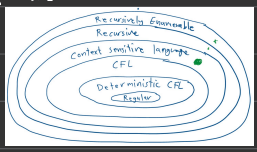
$A \to a$

$B \to b$

$L(G) = \{a^n b^n \mid n \in \mathbb{N}\}$

We define NPDA M s.t.





→ Deterministic PDAs have the following properties:

↳ no ε transitions ↳ δ is a f^n instead of just a relation

↳ end of $x \in \Sigma^*$ is marked by $\dashv$ ↳ closed under $\sim$ by switching accept & non-accept states

Turing Machines and Effective Computability

→ Turing Machines, the most powerful automata, can compute any computable f^n

↳ Among systems that embody effective computability (computable vs non computable)

→ There are natural translations between diff types of data:

there is a bijection from $\{0,1\}^* \to \mathbb{N}$: $x \mapsto \#1(x) - 1$ where $\# y \in \mathbb{N}$ by $y \in \{0,1\}^*$

→ Since all formalisms can be encoded as bitstrings, a formalism can simulate another → computationally equiv.

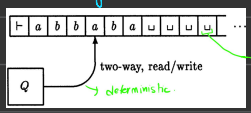
→ Church's Thesis = Declaration that all formalisms are effectively computable in principle ⇒ Turing Machines are programmable

→ One aspect of these systems = Idea of Programs as Data (Programs can encode other programs as data)

→ Universal Simulation = Universal machine U takes on encoded description of another program M and input x & perform simulation of M on x

→ Since there are uncountably many decision problems and only countable TMs ⇒ Naturally uncomputable Problems

→ A Turing Machine = Has a set of states Q, semi-infinite delimited by $\vdash$ and a head moving left and right → Starts at start state s, an input string initially written and head positioned at $\vdash$



Input x is of finite length.

∞ cells to the right

Depending on current state & symbol below the head, writes new symbol on tape, moves head left or right and enters new state

If machine reaches q_+ or q_-, it halts. Otherwise it keeps forever.

q_+ → accept state q_- → reject state

→ Deterministic 1-Tape Turing Machine: $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, q_+, q_-)$ — finite set of states; finite tape alphabet (blank); start $s \in Q$; reject $q_- \in Q$; left end marker; accept $q_+ \in Q$; finite input alphabet

→ We always assume δ satisfies:

$\forall q \in Q \left(\delta(q, \vdash) = (q', \vdash, R) \text{ for some } q' \in Q \right)$

$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ (L: left, R: right)

Ex: Construct a TM that accepts non-CFL $\{a^n b^n c^n \mid n \in \mathbb{N}\}$

$Q = \{s, q_1, \ldots, q_{10}, q_+, q_-\}$

$\Sigma = \{a, b, c\}$

$\Gamma = \Sigma \cup \{\vdash, \sqcup, \dashv\}$

	⊢	a	b	c	⊔	⊣
s	$(s, \vdash, R)$	(s, a, R)	(q_-, b, R)	(q_-, c, R)	$(q_1, \sqcup, L)$	–
q_1	–	$(r_-, -, -)$	(q_1, b, R)	(q_2, c, R)	$(q_3, \sqcup, L)$	–
q_2	–	$(r_-, -, -)$	$(r_-, -, -)$	(q_2, c, R)	$(q_3, \sqcup, L)$	–
q_3	$(s, -, -)$	$(r_-, -, -)$	$(r_-, -, -)$	$(q_4, \sqcup, L)$	$(q_3, \sqcup, L)$	–
q_4	$(r_-, -, -)$	$(r_-, -, -)$	$(q_5, \sqcup, L)$	(q_4, c, L)	$(q_4, \sqcup, L)$	–
q_5	$(r_-, -, -)$	$(q_6, \sqcup, L)$	(q_5, b, L)	–	$(q_5, \sqcup, L)$	–
q_6	$(q_7, \vdash, R)$	(q_6, a, L)	–	–	$(q_6, \sqcup, L)$	–
q_7	–	$(q_8, \sqcup, R)$	$(r_-, -, -)$	$(r_-, -, -)$	$(q_7, \sqcup, R)$	$(r_-, -, -)$
q_8	–	(q_8, a, R)	$(q_9, \sqcup, R)$	$(r_-, -, -)$	$(q_8, \sqcup, R)$	$(r_-, -, -)$
q_9	–	–	(q_9, b, R)	$(q_{10}, \sqcup, R)$	$(q_9, \sqcup, R)$	$(r_-, -, -)$
q_{10}	–	–	–	(q_{10}, c, R)	$(q_{10}, \sqcup, R)$	$(q_+, \sqcup, L)$

Informal: Machine starts at s, then scans to right over input string $y \sqcup \cdots$, $y \in \Gamma^*$ it overwrites it with $\dashv$. Scans left, erasing first c it sees, and same for b & a until $\vdash$. Repeats by going right. If on some pass it says ≥ 1 occurrence of one letter and no occurrences → rejects. If $\vdash$ and $\dashv$ are adjacent, it accepts.

→ At any point, M contains semi-infinite string $y \sqcup^\omega$, $y \in \Gamma^*$

→ Configuration = Global state giving snapshot of all relevant info at some instance

↳ config $\alpha \in Q \times \{y \sqcup^\omega \mid y \in \Gamma^*\} \times \mathbb{N}$

config (p, z, n) — current state; current tape contents; current pos of head $n \in \mathbb{N}$

$\sqcup^\omega$: infinite seq. of blank

z_n where $z \in \Gamma^*$, $n \in \mathbb{Z}^+$ is the n^{th} symbol from z

$S_b^n(z)$ where $n \in \mathbb{Z}^+$, $b \in \Gamma$, $z \in \Gamma^*$ is a string that is the result of substituting z_n with b

→ Start Symbol $x \in \Sigma^*$ $\alpha = (s, \vdash x \sqcup^\omega, 0)$

→ Define Next Relation $\xrightarrow{1}_M$:

$$(p, z, n) \xrightarrow{1}_N \begin{cases} (q, S_b^n(z), n-1) & \text{if } \delta(p, z_n) = (q, b, L) \\ (q, S_b^n(z), n+1) & \text{if } \delta(p, z_n) = (q, b, R) \end{cases}$$

Reflexive Transitive Relation

$\alpha \xrightarrow{0}_M \alpha$, $\alpha \xrightarrow{n+1}_M \beta$ if $\exists \gamma \left(\alpha \xrightarrow{n}_M \gamma \wedge \gamma \xrightarrow{1}_M \beta \right)$

$\alpha \xrightarrow{*}_M \beta$ if $\exists n \in \mathbb{N} \left(\alpha \xrightarrow{n}_M \beta \right)$

→ M accepts $x \in \Sigma^*$ if $(s, \vdash x \sqcup^\omega, 0) \xrightarrow{*}_M (q_+, y, n)$ for some y and n

→ M rejects $x \in \Sigma^*$ if $(s, \vdash x \sqcup^\omega, 0) \xrightarrow{*}_M (q_-, y, n)$ for some y and n

} M halts on x

→ $L(M) = \{x \in \Sigma^* \mid (s, \vdash x \sqcup^\omega, 0) \xrightarrow{*}_M (q_+, y, n) \text{ for some } y \text{ and } n\}$

if you write object of $x \in \Sigma^*$ on tape, M will run & halt at the end w/ $f(x)$ written

→ A Turing Machine is total if $\forall x \in \Sigma^* \left(M \text{ halts on } x \right)$

→ We say α f^n $f : \Omega \to R$ is computable if $\exists$ M : Total TM (M computes it)

→ $A \subseteq \Sigma^*$ is recursively enumerable (r.e) if $\exists$ TM M $(A = L(M))$

→ $A \subseteq \Sigma^*$ is recursive if $\exists$ N : Total TM $(A = L(N))$

→ $A \subseteq \Sigma^*$ is co-recursively enumerable if $\exists$ TM M $(L(M) = \Sigma^* - A = \overline{A})$

→ Decision Problems are decideable if $\exists$ M : Total TM $(f(x) = 1)$

→ Decision Problems are undecideable if $\exists$ M : TM $(f(x) = 1)$

→ Recursive Sets are closed under complement

R.E. sets are not closed under complement

co r.e. set are closed under complement

→ Any TM can be described by a string, so we can give M as input to another machine

→ Halting Problem = Given description of TM M and $x \in \Sigma^*$, decide if M will halt on x → semi-decidable

↳ We can use the fact that halting problem is undecidable to show many other problems are undecidable. Reducing HP to that problem

Ex: Show following is undecidable:

Given TM M, decide if M accepts ε.

Halting Problem Solver (M_1, x):
if Accepts Null (M_2)
return true
return false

$M_2(y)$:
$M_1(x)$
return true

If ∃ Total TM that implements Accepts Null, then (M_1 halts on $x \equiv M_2$ accepts null) so Halting Problem Solver always halts → contradiction

→ Universal Turing Machine $U(M,x) =$ { accept: M accepts x; reject: M rejects x; loop: loops on x } → not total. ⟹ $U'(M,x) =$ { accept: M accepts x; reject: M rejects or loops on x }

→ Claim: We can design such U'

DisProof: For sake of contradiction, assume we have such U. We can create $\bar{U}$ to complement U' s.t. $\bar{U}(M)$:
if $U'(M,M)$: loop forever
else: accept

$\bar{U}(\bar{U})$ → $\bar{U}$ accepts $\bar{U}$ → $\bar{U}$ rejects $\bar{U}$ x
$\bar{U}$ loops $\bar{U}$ x
$\bar{U}$ rejects $\bar{U}$ → $\bar{U}$ accepts $\bar{U}$ x
contradicts

A) Easily decidable since $|Q|$ of M can be read off from the encoding of M → short simulation

B) Decidable since we can simulate M with ε with U for 481 steps & accept or reject on whether M halted by that time

c) Decidable since we can simulate M on all input lengths ≤ 481 for 481 steps. If M takes > 481 steps for some x, it would take > 481 steps on some input of length ≤ 481.

d) Decidable since if M takes > 481 steps for all x with $|x| \leq 481$ it will take > 481 steps for all x

e) For problem (e), if M never moves more than 481 tape cells away from the left endmarker, then it will either halt or loop in such a way that we can detect the looping after a finite time. This is because if M has k states and m tape symbols, and never moves more than 481 tape cells away from the left endmarker, then there are only $482km^{481}$ configurations it could possibly ever be in, one for each choice of head position, state, and tape contents that fit within 481 tape cells. If it runs for any longer than that without moving more than 481 tape cells away from the left endmarker, then it must be in a loop, because it must have repeated a configuration. This can be detected by a machine that simulates M, counting the number of steps M takes on a separate track and declaring M to be in a loop if the bound of $482km^{481}$ steps is ever exceeded.

Decideable & Undecideable Problems:

→ Some decision problems involving Turing Machines:

(a) has at least 481 states?
(b) takes more than 481 steps on input ε?
(c) takes more than 481 steps on *some* input?
(d) takes more than 481 steps on *all* inputs?
(e) ever moves its head more than 481 tape cells away from the left endmarker on input ε?
(f) accepts the null string ε?
(g) accepts any string at all?
(h) accepts every string?
(i) accepts a finite set?
(j) accepts a regular set?
(k) accepts a CFL?
(l) accepts a recursive set?

a) through e) are decidable
f) through l) are undecidable

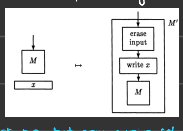
Shown Using: decision procedure from the problems → decision procedure for halting problem
Reduction

Best way to show Problem is decideable: Since a total TM that accepts exactly the "yes" instances & rejects "no" instances while not looping

f) We will show that it is undecideable whether machine accepts ε since ability to decide would give ability to decide HP

Suppose we could decide if given machine accepts ε. Say we are given TM M and $x \in \Sigma^*$ and we wish to determine if M halts on x

Construct M' that does the following on y:
i) erases y
ii) writes x on its tape
iii) runs M on x
iv) accepts if M halts on x



Note that M' does same thing on all y inputs

∴ $\forall y \in \Sigma^*$. $L(M') =$ { Σ^* if M halts on x; $\emptyset$ if M loops on x }

Apply this procedure to if given machine accepts ε ⇒ tell us whether M halts on x.

Since HP is undecideable ⇒ undecideable whether given machine accepts ε

→ To show j), k) and l) are undecideable, pick r.e. but nonrecursive set A and modify construction as follows: Given M and x build M''

i) saves y on separate track of its tape
ii) writes x on a different track
iii) runs M on x
iv) if M halts on x, M'' runs machine accepting A on eg. y and accepts if that machine accepts.

∴ $L(M'') =$ { A if M halts on x; $\emptyset$ if M loops on x }

since A is not recursive(& regular), $\emptyset$ is all three