Equivalence of DFA and NFA:

**Definition:**

\[ M_1 \equiv M_2 \text{ if } L(M_1) = L(M_2) \]

Let \[ L(M_1) = L(M_2) \]

\[ \{ (10)^n : n \geq 0 \} \]

DFA is a restricted kind of NFA.

\[ L(DFA) \subseteq L(NFA) \]

We can also convert an NFA into a DFA.

The technique is:

For an NFA \( \delta \), \( \delta^*(q_i, \omega) = \{ q_j : j \in S \} \)

create a new state \( \{ q_j : j \in S \} \) for all \( q_i \).

**Example:** Convert to a DFA

1) Initial state of DFA will be \( \{ q_0 \} \) too.
2) After reading a, NFA will be in \( \{ q_1, q_2 \} \)
   \( \delta_\text{NFA}(\{ q_0, q_3 \}, a) = \{ q_1, q_2 \} \)
   Call \( \{ q_1, q_2 \} \) another state for the DFA.
3) \( \delta_\text{DFA}(\{ q_0, q_3 \}, b) = \emptyset \), call this another state also
   make this a non-final trap state.
4) Study NFA for \( \{ q_1, q_2 \} \) state, ... we see
   \( \delta_\text{NFA}(\{ q_1, q_2 \}, a) = \{ q_1, q_2 \} \) and
   \( \delta_\text{NFA}(\{ q_1, q_2 \}, b) = \{ q_3 \} \)

DFA:

\[ \{ q_0 \} \]

\[ \{ q_1, q_2 \} \]

\( \delta_\text{DFA}(q_0, a) = \{ q_1, q_2 \} \)

\( \delta_\text{DFA}(q_0, b) = \emptyset \)

\( \delta_\text{DFA}(q_1, a) = \{ q_1, q_2 \} \)

\( \delta_\text{DFA}(q_1, b) = \{ q_3 \} \)

\( \delta_\text{DFA}(q_2, a) = \{ q_1, q_2 \} \)

\( \delta_\text{DFA}(q_2, b) = \{ q_3 \} \)

\( \delta_\text{DFA}(q_3, a) = \emptyset \)

\( \delta_\text{DFA}(q_3, b) = \{ q_3 \} \)
Deterministic Finite Automata (DFA)

Input alphabet \( \{0, 1\} \)

DFA defined by \( M = (Q, \Sigma, \delta, q_0, F) \)

\[ Q = \{q_0, q_1, q_2\} \] initial states

\[ \Sigma = \{0, 1\} \] input alphabet

\[ \delta : Q \times \Sigma \to Q \] transition function

\[ \delta(q_0, 0) = q_0 \]

\[ \delta(q_0, 1) = q_1 \]

\[ \delta(q_1, 0) = q_1 \]

\[ \delta(q_1, 1) = q_2 \]

\[ \delta(q_2, 0) = q_2 \]

\[ \delta(q_2, 1) = q_3 \]

\[ F = \{q_3\} \] final state set

This DFA accepts strings 01, 101, 0111, 1100, etc.

doesn't accept 00, 100, 1100, etc.

Extended transition function

\[ \delta^* : Q \times \Sigma^* \to Q \]

\[ \delta^*(q_0, 1) = q_0 \]

\[ \delta^*(q_1, 10) = q_2 \]

\[ \delta^*(q_2, 100) = q_3 \]

\[ \delta^*(q_3, 0) = q_3 \]

Recursive definition:

\[ \delta^*(q, \varepsilon) = q \]

\[ \delta^*(q, w) = \delta(\delta^*(q, w), a) \]

Language accepted by DFA \( M \) is the set of strings accepted.

\[ L(M) = \{ w \in \Sigma^* : \delta^*(q_0, w) \in F \} \]
Regular language a language s.t. 
\[ L = L(M) \] accepted by a DFA \( M \).

Non deterministic finite accepter (NFA) 

\[ \mathcal{N} = (Q, \Sigma, \delta, q_0, F) \]

\[ \delta: Q \times (\Sigma \cup \{ \epsilon \}) \rightarrow 2^Q \]

\[ \delta(q_0, a) = \{ q_1, q_2 \} \]

Extended transition function is a set

\[ \delta^*(q_0, w) = \varnothing \]

\[ \delta^*(q_1, a) = \{ q_0, q_1, q_2 \} \]

Language accepted by NFA

If there is a "walk" labeling \( w \) then it \( N \) to

final state

\[ L(M) = \{ w \in \Sigma^* : \delta^*(q_0, w) \cap F \neq \varnothing \} \]

NFA can also have a state with no outgoing edge for some input (dead configuration)

e.g., we might have

\[ \delta^*(q_0, 110) = \varnothing \]

Why non-determinism?

ex. (a) For backtracking (car in a maze game)

\[ L(\overline{M}) = L(M) \cup L(M^\epsilon) \]

ex. (b) effectively, DFA and NFA are equivalent, so for ease whenever possible.
with state for DFA

\[ Q_0 = \{ q_0, q_2 \} \]

\[ \delta (q_0, a) = \{ q_1, q_2 \} \]

\[ \delta (q_2, b) = \phi \]

\[ \delta (q_2, a) = \{ q_1, q_2 \} \cup \delta (q_2, a) \]

\[ = \{ q_1, q_2 \} \cup \{ q_2 \} \]

\[ = \{ q_0, q_2 \} \]

\[ \delta (q_0, a) = \{ q_1, q_2 \} \]

\[ = \{ q_2 \} \cup \{ q_2 \} \]

\[ = \{ q_2 \} \cup \{ q_2 \} = \{ q_0 \} \]
Reduction of the \# of states in F.A.

Any D.F.A. defines a unique language but the converse is not true. You can have more than one D.F.A. for the same language, we need for state reduction (for simplicity and efficiency).

Definition: Two states \( p \) and \( q \) of a D.F.A. are indistinguishable if

\[
\delta^*(p, w) \in F \implies \delta^*(q, w) \in F
\]

and

\[
\delta^*(p, w) \notin F \implies \delta^*(q, w) \notin F
\]

\[\forall w \in \Sigma^* \]

\[q \iff \exists w \in \Sigma^* \text{ s.t.} \]

\[
\delta^*(p, w) \in F \text{ and } \delta^*(q, w) \notin F \text{ or vice versa}
\]

New \( p \equiv q \) are distinguishable by \( w \).

Indistinguishability is an equivalence relation i.e.

If \( \delta(p, x) = \delta(q, x) \) then \( p \equiv q \) for \( x \in \Sigma \).

Reduce states by finding + combining I states.

Algorithm (Mark), 1) remove all unaccessible states (exhaustive approach)

2) Consider all pairs of states \( \langle p, q \rangle \), if \( p \not\in F \) and \( q \not\in F \)

(or vice versa) mark the pair \( \langle p, q \rangle \) as distinguishable.

3) Repeat till no previously unmarked pairs are marked.

(\( \delta \) (p, q) true and all \( a \in \Sigma \), compute \( \delta(p, a) = \delta(q, a) \).

\( \delta(p, a) = \delta(q, a) \).

\( \delta(p, a) \) i.e. the pair \( \langle p, q \rangle \) is marked as distinguishable then mark \( \langle p, q \rangle \) as distinguishable.

Reduce -- partition the state set \( Q \) into disjoint subsets \( \{B_i, B_2, \ldots, B_k\}, i \in \{1, 2, \ldots, k\}, f \not\in B_i \) occurs in only one set + elements in each set are indistinguishable + any 2 elements from any set are distinguishable.

(exercise 11)

Given D.F.A. \( M = (Q, \Sigma, \delta, s_0, F) \) create \( M' = (Q', \Sigma, \delta', s_0', F') \)
1. For each set \( \{8_i, 9_i, \ldots, 9_k\} \) create a state labeled \( c_{ij} \) for \( M \).
2. For each \( \delta(8_i, a) = 8_p \) of \( M \), find sets to which \( 8_i \) and \( 8_p \) belong and add:
   \[ \hat{\delta}(c_{ij} \cdot k, a) = \text{lem} \ldots n \quad \text{if} \quad 8_i \in \{8_i, 9_i, \ldots, 9_k\} \quad 8_p \in \{8_i, 9_i, \ldots, 9_k\} \]
3. \( \hat{s}_0 \) is the state of \( \hat{M} \) that includes 0.
4. \( \hat{F} \) is the set of all states whose label contains \( c \) s.t.
   \[ \hat{s}_0 \in \hat{F} \]

\[ \hat{M} \text{ is minimal s.t.} \]

\[ L(\hat{M}) = L(M) \]

and \( \hat{M} \) contains the minimum number of states.
Regular Expressions

1. Let \( \Sigma \) be a given alphabet, then
2. \( \emptyset, \Sigma, \text{ and } a \in \Sigma \) are all regular expressions (R.E)
3. If \( \alpha \) and \( \beta \) are R.E, then \( \alpha + \beta, \alpha \beta, \alpha^* \) and \( \alpha^\dagger \) are R.E
4. A string is a R.E if derivable by finite application of 2).
5. Show \((a+b.c)^* . (c+\emptyset)\) for \( \Sigma = \{a,b,c\} \) R.E

Language of Regular Expressions

- \( L(\emptyset) = \emptyset \) is the empty set.
- \( L(\Sigma) \) is the set of all symbols in \( \Sigma \).
- \( L(a) = \{a\} \) is a singleton set.
- \( L(\emptyset) \) is the empty set.
- \( L(\Sigma) \) is the set of all symbols in \( \Sigma \).
- \( L(a^*) \) is the set of all strings over \( \Sigma \) that can be formed using the symbol \( a \).
- \( L(a^\dagger) \) is the set of all strings over \( \Sigma \) that can be formed using the symbol \( a \).

Regular Expressions => Regular Languages

If \( R \) is a regular expression, then \( \exists \) some NFA for \( L(R) \) such that \( L(R) \) is a regular language.
Find NFA accepting $L(x)$ for $x = \text{a} + \text{b}(\text{a} + \text{b})^* (\text{a}^* + \text{a})$

$M_1$ for $L(a+b)$

$M_2$ for $L(\text{b}a^*+\text{a})$

$M$ for $L((a+b)(\text{b}a^*+\text{a}))$

Regular expressions for regular languages. ($R \in \Sigma^* \iff L(R)$)

**Regular Expressions for Regular Languages:**

$R \in \Sigma^*$

$R = a^* + a(a+b)c^*$

**General transition graph**

- Removing a non-starting and non-final state

$R = \text{a} \cdot \text{e} \cdot \text{d} \cdot \text{c} \cdot \text{e} \cdot \text{b}$

- General and use with

$L_1 = \lambda_1 \text{a} \cdot \lambda_2 \text{a}^* (\lambda_4 + \lambda_3 \lambda_1 \lambda_2)^*$

$R = (b + \text{a} \cdot \text{b} \cdot \text{a}^*) \text{a} \cdot \text{b} \cdot \text{b} (\text{a} \cdot \text{b})^*$

Regular expression for some pattern e.g. phone numbers, email addresses, etc.
Regular Grammars

- Grammar $G = (V, T, S, P)$ is right-linear if all rules have the form:
  
  $A \rightarrow xB$
  
  $A \rightarrow x$
  
  $A, B \in V$, $x \in T^*$

- Left-linear

  $A \rightarrow Bx$
  
  $A \rightarrow x$

- Regular grammar is either right-linear or left-linear

- Linear grammar is:
  
  $S \rightarrow A$
  
  $A \rightarrow aB|\lambda$
  
  $B \rightarrow A$

Right-linear grammars generate regular languages \textit{d v i e r v a i n a}

Production rule $D \rightarrow DE$

- $ab \cdots CD \Rightarrow ab \cdots cdE$

- $ab$ by left-linear grammar