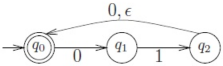
CSC236H
Introduction to the Theory of Computation

A **Nondeterministic Finite Automaton (NFA)** $\mathcal{N}$ is a quintuple $\mathcal{N} = \langle Q, \Sigma, \delta, q_0, F \rangle$ where:

- Q is the **set of states** in $\mathcal{N}$;
- Σ is the **alphabet** of symbols used by $\mathcal{N}$;
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$ is the **transition function**;
- $q_0 \in Q$ is the **initial state** of $\mathcal{N}$;
- $F \subseteq Q$ is the set of **accepting states** of $\mathcal{N}$.

- A string $w \in \Sigma^*$ is **accepted** by $\mathcal{N}$, if and only if at least one of the possible states in which the automaton could be after processing input w is an accepting state.

- The language **accepted** (or **recognised**) by an NFA $\mathcal{N}$, denoted $\mathcal{L}(\mathcal{N})$, is the set of all strings accepted by $\mathcal{N}$.


0, ϵ
q0
0
q1
1
q2

Equivalence of the Three Representations of Regular Languages

Theorem. Let L be a language. The following three statements are equivalent:

1. There is a regular expression that denotes L.
2. There is a DFA that accepts L.
3. There is an NFA that accepts L.

Proof. (Optional) See **Lemma 7.18**, **Theorems 7.22**, and **Theorems 7.23** in the Course Notes.

Definition. A language is **regular** if and only if it is denoted by some **regular expression**; or, equivalently, if and only if it is accepted by a **DFA**, or, equivalently, if and only if it is accepted by an **NFA**.

Why NFA's?

- There are languages that can be accepted by an NFA that are much smaller than the smallest DFA that accepts the same language.

- Conceptual simplicity of NFA's.

Example: Let $L = \{x \in \{0,1\}^* : x = y1z,$ for some $y, z \in \{0,1\}^*$ s.t. $|z| = 3\}$.
That is, L consists of all strings with at least 4 symbols, where the 4th symbol from the end is 1.

- The smallest DFA that accepts L has **16** states.

- There's an NFA with **5** state which accepts L.

Converting NFA's to DFA's (Optional)

- There is an algorithm (**subset construction**) for converting an NFA $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ to a DFA $\hat{\mathcal{M}} = (\hat{Q}, \Sigma, \hat{\delta}, \hat{q_0}, \hat{F})$ that accepts the same language as $\mathcal{M}$.

  1. $\hat{Q} = \mathcal{P}(Q)$.

  2. $\hat{q_0}$: the set of all states reachable from the initial state of $\mathcal{M}$ via ϵ-transitions.

  3. $\hat{F}$: all states that contain an accepting state of $\mathcal{M}$.

  4. Let $a \in \Sigma$, $\hat{q} \in \hat{Q}$, and suppose $\hat{q}$ is associated with $\{q_1, ..., q_n\} \in \mathcal{P}(Q)$.
     To compute $\hat{\delta}(\hat{q}, a)$, identify all states in Q that can be reached from $\{q_1, ..., q_n\}$ by reading a:
     - Let $r = \{\}$.
     - Do the following steps for each $q_i \in \{q_1, ..., q_n\}$:

       – $r = r \cup \delta(q_i, a)$.

       – for each $q' \in \delta(q_i, a)$ identify the set of states which can be reach by a ϵ-transition, call it $\mathcal{E}(q')$. Then, $r = r \cup \mathcal{E}(q')$.

Lemma 7.18 in the Course Notes.

- Find a regular expression for the given language.
- Draw NFA's for sub-expressions that do not include $*$.
- Draw NFA's for sub-expressions contain $*$.
- Connect the NFA's corresponding to the sub-expressions.
- Might need to add some ϵ-transitions to make sure all strings in the given language are accepted.

IMPORTANT: the above steps are not a generic procedure for designing NFA's.

They only provide some guidelines which make designing NFA's easier.

Tips for Designing NFA's – Example

Let $L = \{x \in \{0,1\}^* : x = y1z, \text{ for some } y, z \in \{0,1\}^* \text{ s.t. } |z| = 3\}$.

That is, L consists of all strings with at least 4 symbols, where the 4th symbol from the end is 1.

Give an NFA which accepts L.

Tips for Designing NFA's – Example

$L = \{x \in \{0,1\}^* :$ x contains some substring of length 4 whose first and last characters are the same$\}$.

Give an NFA which accepts L.

Definition. A language is **regular** if and only if it is denoted by some **regular expression**; or, equivalently, if and only if it is accepted by a **DFA**, or, equivalently, if and only if it is accepted by an **NFA**.

- When is a language **not regular**?
  - DFA's/NFA's have fixed, finite states (memory).
  - If recognizing a language requires unfixed or unlimited memory (states), it cannot be represented by any DFA/NFA. Hence it's not regular.

Examples: $L = \{0^n 1^n : n \in \mathbb{N}\} = \{\epsilon, 01, 0011, 000111, ...\}$ is **not** regular.
For a contradiction, assume that L is regular. Then, there is a DFA that accepts L.
Suppose the DFA has k states.
Now, consider the behaviour of the DFA on input string $0^{k+1}1^{k+1}$:

Theorem. Regular languages are **closed** under **complementation**, **union**, **intersection**, **concatenation** and the **Kleene star** operation.
That is, if L and L' are two regular languages, then so are all of the following:
$\bar{L}, L \cup L', L \cap L', L \circ L', L^{\circledast}$.

Proof:
(Proof Idea: Consider the DFA's/NFA's/Regular Expressions that represent L and L'.
For each of the above five operations, show that a DFA/NFA/Regular Expression can be constructed based on the DFA's/NFA's/Regular Expressions representing L and L'.)
Prove that if L is regular then $L' = \{xy : x \in L \text{ and } y \notin L\}$ is also regular.

Let $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ and $\mathcal{M}' = (Q', \Gamma, \delta', q_0', F')$ be DFA's that accept L and L' respectively.

Proving Regularity by Closure Properties

If L is a regular language over $\Sigma = \{a, b\}$, show that language $L' = \{w : w = x1 \text{ for some } x \in L\}$ is also regular.

Proof Idea: Use the DFA for L to construct a DFA for L'.

Proving Non-Regularity by Closure Properties

Disprove the following statement:

If $L_1 \cup L_2$ is regular and L_1 is regular, then L_2 is regular.