CS372
FORMAL LANGUAGES & THE THEORY OF COMPUTATION

Dr. Nguyen Thi Thu Huong
Phone: +84 903253796
Email: huongnt@soict.hust.edu.vn,
huong.nguyenthithu@hust.edu.vn

Unit 2
Finite Automata, Nondeterminism

Introduction

- Automata theory is a wide applicable area in Computer Science.
- Automata Theory is a part of our life.
- Have you ever seen a vendor machine , or any equipment controlled by an automaton like a washing machine,a traffic light, an elevator, etc ?

Opinions

- One uses, the Automata Theory in daily life.
- If we know automata, we can understand machines languages.
- Through Automata Theory one can learn software, algorithm, and hardware basics.
- You are modeling some systems.

Modeling a finite automaton

- In Automata Theory, you can build your own world and you can choose your rules.

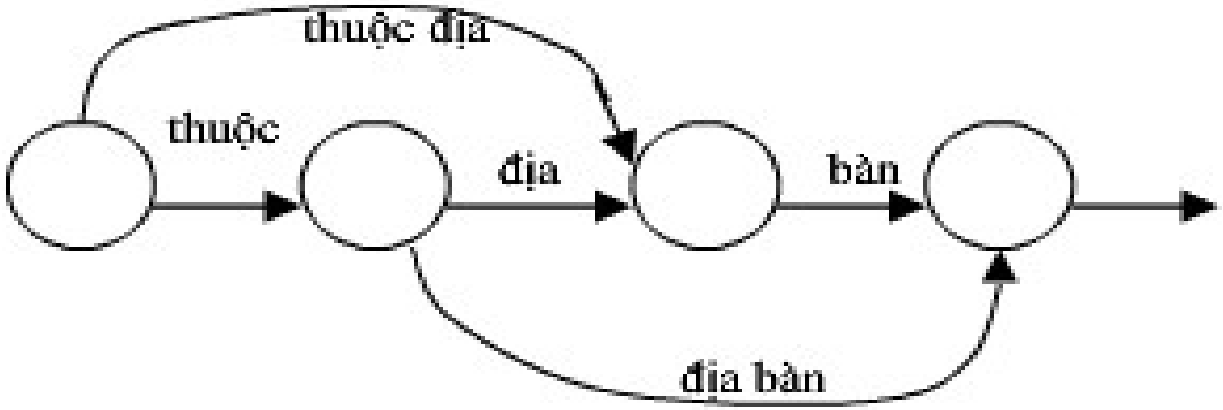
- Automata Theory makes some relations to hardware with software.

recognizers vs transducer

- There exist **several types of finite-state automata**, which can be divided into two main categories:
  - **recognizers**: either accept the input or do not
  - **transducers**: generate output from given input

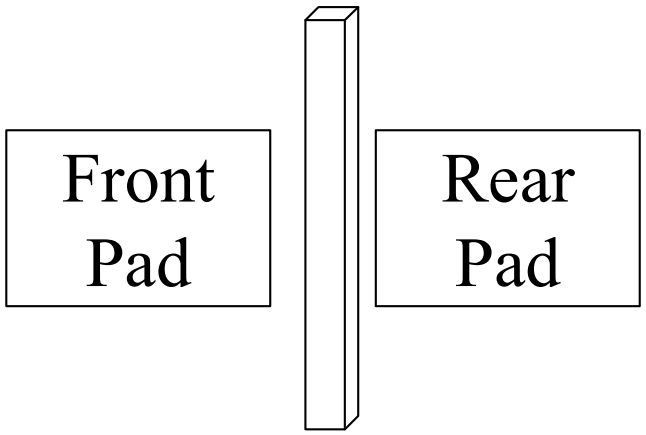
A recognizer example

- A branch of an Automaton for Vietnamese segmentation
- Express the relationship of morphemes in Vietnamese phrase “thuộc địa bàn”



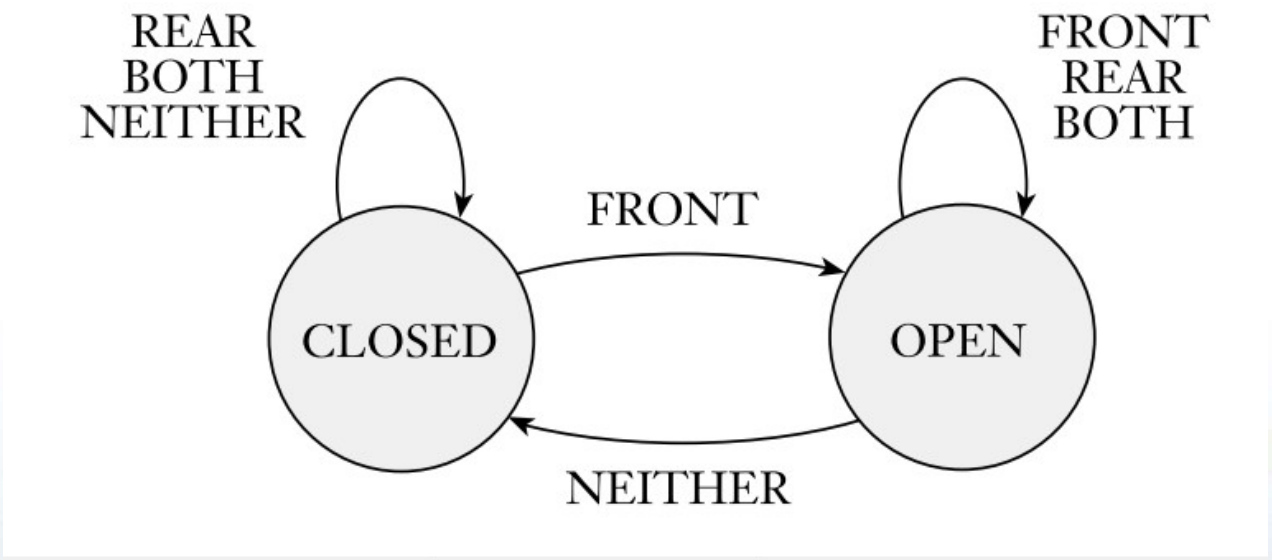
Simple Example – 1 way door

- Consider a one-way automatic door.
- This door has two pads that can sense when someone is standing on them, a front and rear pad.
- We want people to walk through the front and toward the rear, but not allow someone to walk the other direction:



One Way Door

- Let's assign the codes to our different input cases:
  - Nobody on either pad
  - Person on front pad
  - Person on rear pad
  - Person on front and rear pad
- We can design an automaton so that the door doesn't open if someone is still on the rear pad and hit them:

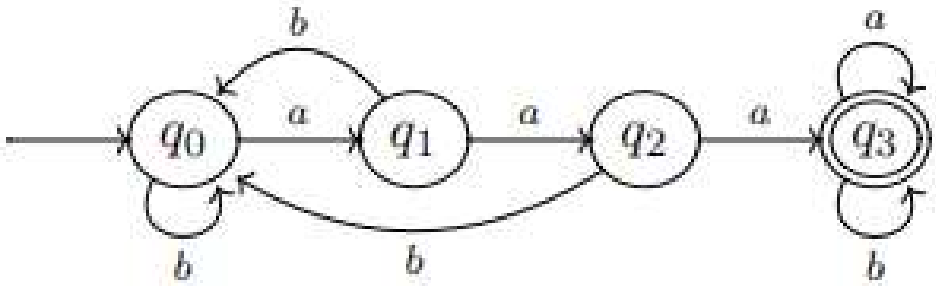


State diagrams of finite automata

- State diagrams are directed graphs whose nodes are states and whose arcs are labeled by one or more symbols from some alphabet Σ.
- One state is initial (denoted by a short incoming arrow)
- Several states are final/accepting (denoted by a double circle).
- DFA: For every symbol $a \in \Sigma$ there is an arc labeled a emanating from every state

Example

- State diagram of a DFA accepts all strings over {a,b} that have 3 consecutive a's.



- Consider 2 strings: abaab and baaab
  - List of states for abaab: $q_0 \Rightarrow q_1 \Rightarrow q_0 \Rightarrow q_1 \Rightarrow q_2 \Rightarrow q_0$: string is not accepted
  - List of states for baaab: $q_0 \Rightarrow q_0 \Rightarrow q_1 \Rightarrow q_2 \Rightarrow q_3 \Rightarrow q_3$: string is accepted

Formal Definition of a Deterministic Finite Automaton (DFA)

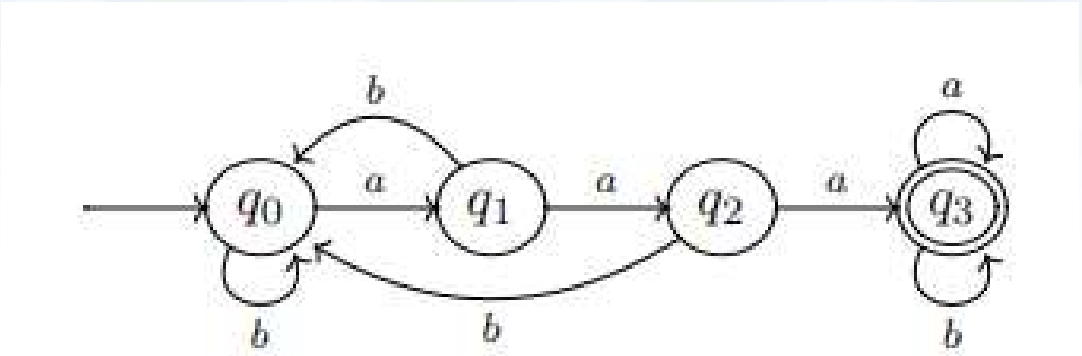
A DFA is represented as the five-tuple: $M = (Q, \sum, \delta, q_0, F)$ where

1. Q is a finite set of states,
2. $\sum$ is the alphabet of input symbols,
3. $q_0 \in Q$ is the start/initial state,
4. $F \subseteq Q$ Set of final states
5. $\delta: Q \times \Sigma \rightarrow Q$ is a transition function.

This function

- Takes a state and input symbol as arguments.
- Returns a state.
- One "rule" would be written $\delta(q, a) = p$, where q and p are states, and a is an input symbol.
- Intuitively: if the DFA is in state q, and input a is received, then the DFA goes to state p (note: q = p OK).

DFA Example D_1



- Recognize set of all strings over{a,b} contain 3 consecutive a's
- Formal definiton

$M = (Q, \Sigma, \delta, q_0, F)$

$Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{a,b\}$, $F = \{q_3\}$

δ	a	b
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_3	q_0
q_3	q_3	q_3

Configuration

- Used in formal description of DFA's computation.
- Definition: Let M = $(Q, \sum, \delta, q_0, F)$ be a DFA. We say that a word qx, $q \in Q$, $x \in \Sigma^*$, is a configuration of M. It represents the current state of M and the remaining unread input of M.
- A configuration of a DFA,M, contains all the information necessary to continue M's computation.
- In programming parlance it is equivalent to a dump of the current value of all variables of a program and the current position in the program.

Language accepted by a DFA

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA and let $w = w_1 w_2 \ldots w_n$ be a string where each w_i is a member of alphabet Σ.
- M accepts w if a sequence of states $r_0, r_1, \ldots, r_n$ in Q exists with three conditions:
  1. $r_0 = q_0$ (the first configuration is $q_0 w$)
  2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for I = 0, … , n-1
  3. $r_n \in F$ (the last configuration is r_n)
- Language recognized by DFA M:

  $L(M) = \{w \mid w \in \Sigma^* \; q_0 w \Rightarrow^* q \in F\}$

  (In other words, the language is all of those strings that are accepted by the finite automata).

Nondeterminism

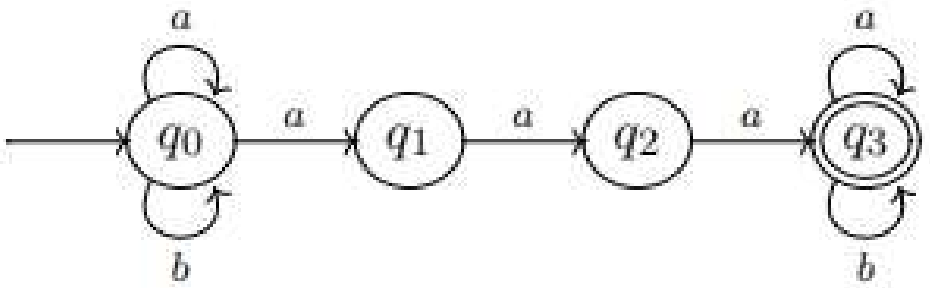
- Nondeterministic Finite Automata
- NFA with ε- transition

Nondeterministic Finite Automata

- A NFA (nondeterministic finite automaton) is able to be in several states at once.
  - In a DFA, we can only take a transition to a single deterministic state
  - In a NFA we can accept multiple destination states for the same input.
  - Another way to think of the NFA is that it travels all possible paths, and so it remains in many states at once. As long as at least one of the paths results in an accepting state, the NFA accepts the input.
- NFA is a useful tool
  - More expressive than a DFA.
  - BUT we will see that it is **not more powerful**! For any NFA we can construct an equivalent DFA

NFA Example: N_1

- NFA accepts set of all strings over {a,b} contain 3 consecutive a's
- N_1 includes 4 states like D_1



Epsilon Transition

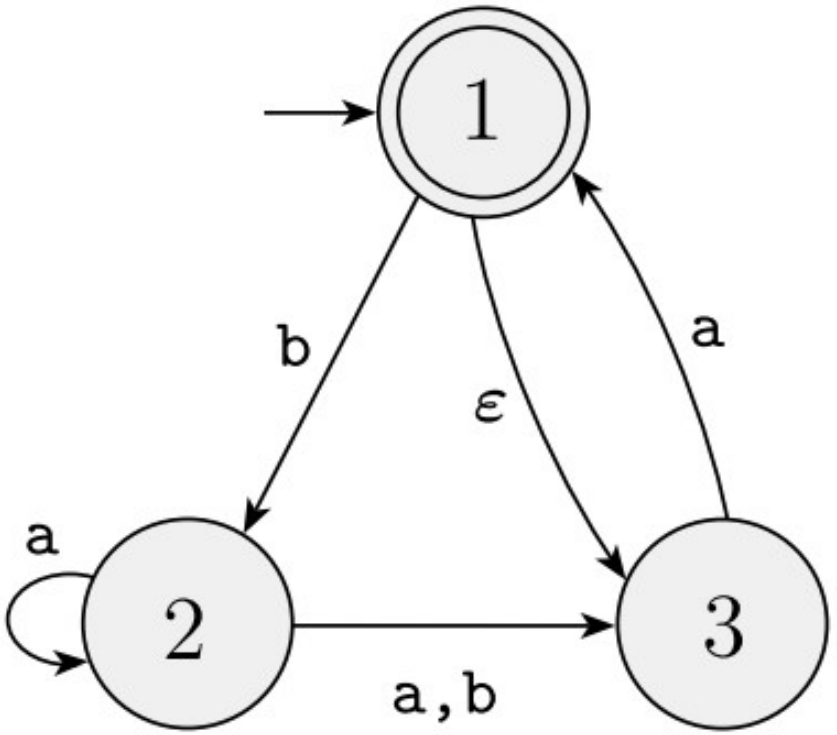
- Transition function δ is now a function that takes as arguments:
  - A state in Q and
  - A member of $\sum \cup \{\varepsilon\}$; that is, an input symbol or the symbol ε.Note that ε is not a symbol of the alphabet $\sum$.

Another NFA Example: N_2

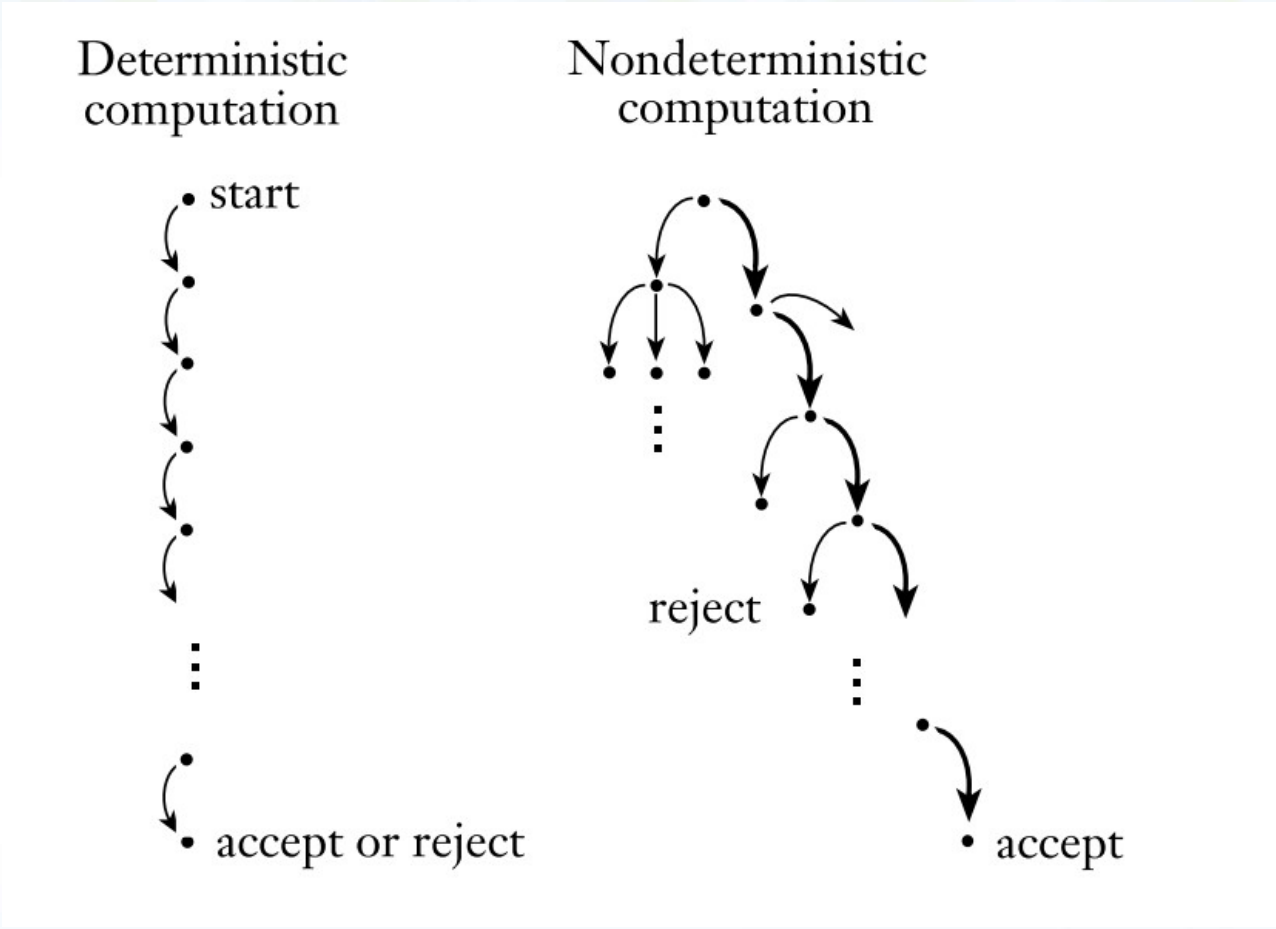
In this NFA, the string baaa is accepted by the 2 paths 12231 and 123131.

In path 12231:12 matches b, 22 matches a, 23 matches a, 31 matches a.

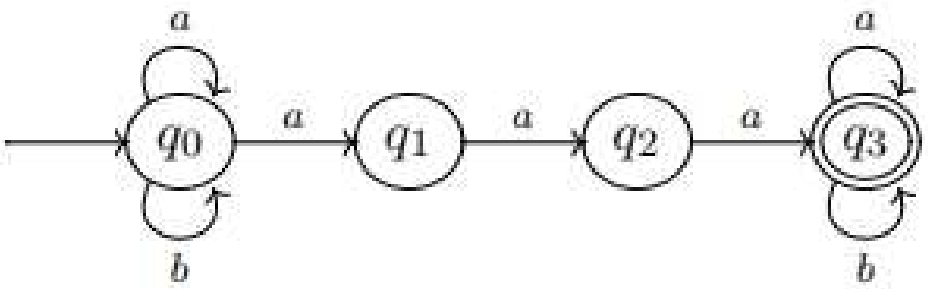
In path 123131:12 matches b, 23 matches a, 31matches a, 13 matches ε, 31 matches a. In other words, the accepted string is baaεa.

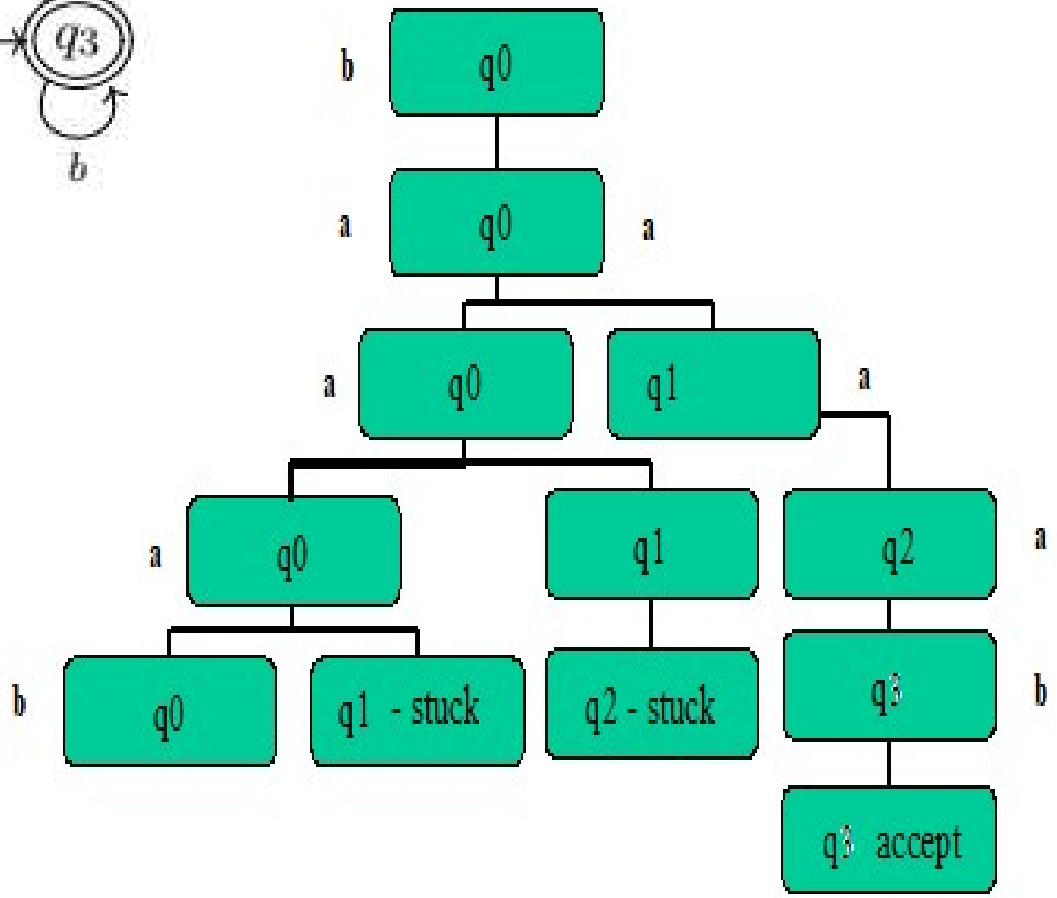


Determinism vs Nondeterminism



Computation of N_1 on baaab





Formal Definition of NFA

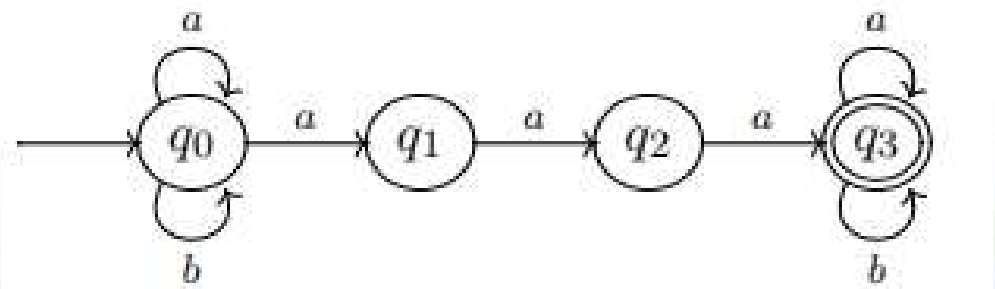
- Definition: An ε-NFA is represented as a five – tuple $(Q, \sum, \delta, q_0, F)$ where
  1. Q is a finite set of states,
  2. $\sum$ is the alphabet of input symbols,
  3. $q_0 \in Q$ is the start/initial state,
  4. $F \subseteq Q$ Set of final states
  5. $\delta: Q \times \{\Sigma \cup \{\varepsilon\}\} \rightarrow P(Q)$

Formal Notation of NFA N_1

$N_1 = (Q, \Sigma, \delta, q_0, F)$

$Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{a,b\}$, $F = \{q_3\}$

δ	a	b
q_0	$\{q_0,q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	$\varnothing$
q_2	$\{q_3\}$	$\varnothing$
q_3	$\{q_3\}$	$\{q_3\}$

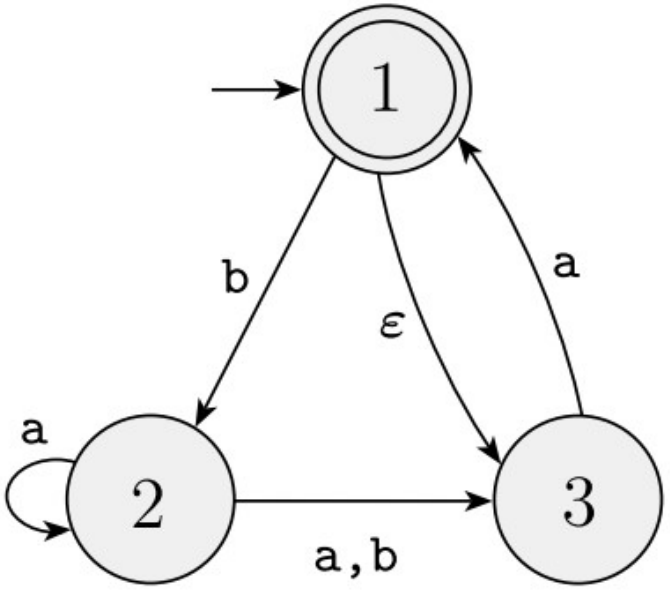


Formal Notation of NFA N_2

$N_1 = (Q, \Sigma, \delta, q_0, F)$

$Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{a,b\}$, $F = \{q_3\}$

δ	a	b	ε
1	$\varnothing$	{2}	{3}
2	{2, 3}	{3}	$\varnothing$
3	{1}	$\varnothing$	$\varnothing$



Language accepted by an NFA

- Same idea as the DFA
- Let N = $(Q, \sum, \delta, q_0, F)$ be an NFA and let w = $w_1 w_2 \ldots w_n$ be a string where each w_i is a member of alphabet $\sum$.
- N **accepts** w if a sequence of states $r_0 r_1 \ldots r_n$ in Q exists with three conditions:
  1. $r_0 = q_0$
  2. $r_{i+1} \in \delta(r_i, w_{i+1})$ for i=0, … , n-1
  3. $r_n \in F$

Observe that $\delta(r_i, w_{i+1})$ is the set of allowable next states

We say that N recognizes language A if A = {w | N accepts w }

Equivalence of DFA's and NFA's

- For most languages, NFA's are easier to construct than DFA's
- But it turns out we can build a corresponding DFA for any NFA
  - The downside is there may be up to 2^n states in turning a NFA into a DFA. However, for most problems the number of states is approximately equivalent.
- Theorem: A language L is accepted by some DFA if and only if L is accepted by some NFA;
- i.e. : L(DFA) = L(NFA) for an appropriately constructed DFA from an NFA.
  - Informal Proof: It is trivial to turn a DFA into an NFA (a DFA is already an NFA without nondeterminism). The following slides will show how to construct a DFA from an NFA.

NFA to DFA : Subset Construction

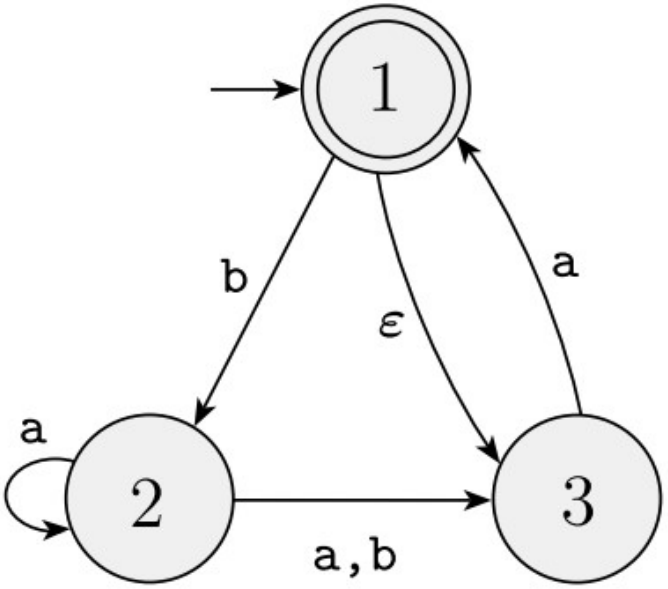
- Let an NFA N be defined as $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$
- The equivalent DFA $D = (Q_D, \Sigma, \delta_D, q'_0, F_D)$, where

  $Q_D = P(Q_N)$; i.e. Q_D is the set of all subsets of Q_N (the power set of Q_N). Often, not all of these states are accessible from the start state; these states may be "thrown away"
- F_D is the set of subsets S of Q_N such that $S \cap F_N \neq \emptyset$. That is, F_D is all sets of N's states that include at least one accepting state of N.
- For each set $S \subseteq Q_N$ and for each input symbol a in Σ

$$\delta_D(S, a) = \bigcup_{p \in S} \delta_N(p, a)$$

- That is, to compute $\delta_D(S, a)$ we look at all the states p in S, see what states of N goes to starting from p on input a, and take the union of all those states.

Subset Construction Example

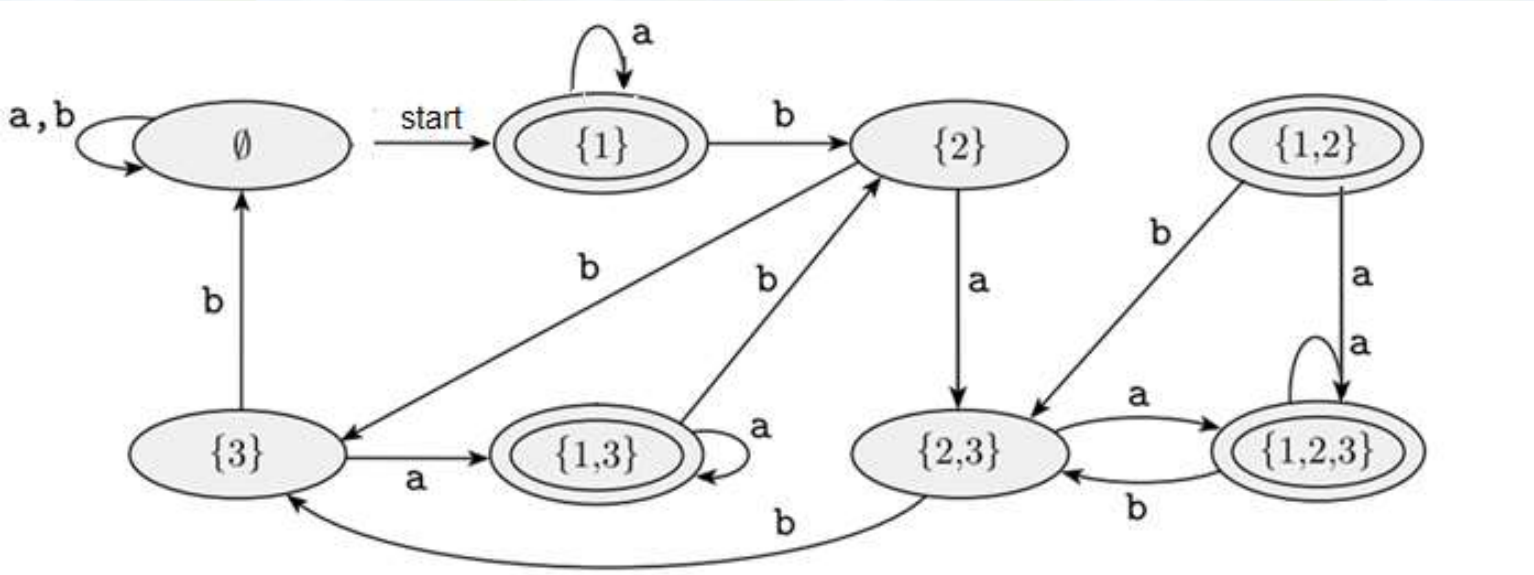


- Consider the NFA N_2:
- The power set of these states is: {Ø, {1},{2}, {3}, {1,2}, {1,3}, {2,3}, {1,2,3}}

	a	b
Ø	Ø	Ø
→{1}	{1}	{2}
{2}	{2,3}	{3}
{3}	{1,3}	Ø
{1,2}	{1,2,3}	{2,3}
{1,3}	{1.3}	{2}
{2,3}	{1,2,3}	{3}
{1,2,3}	{1,2,3}	{2,3}

- New transition function with all of these states and go to the set of possible inputs.

DFA D_4 (equivalent to N_4)



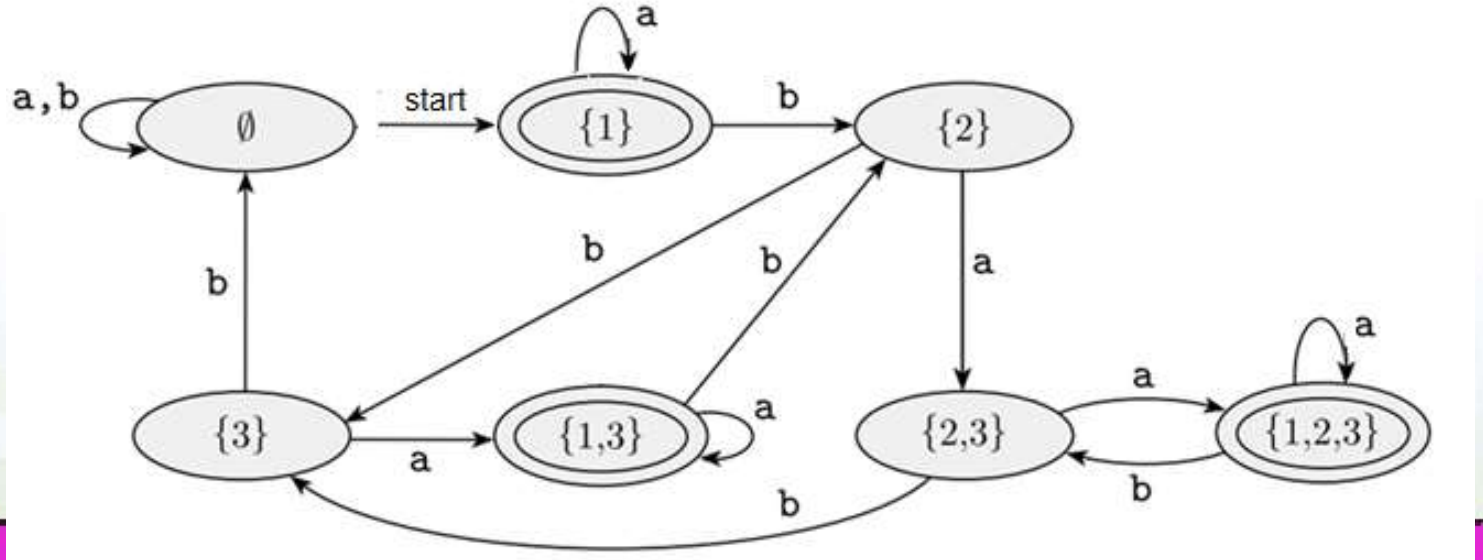
Observe that there is no path from state {1} to state {1,2} so {1,2} is remove from N_4

Subset Construction (continued)

Many states may be unreachable from our start state. A good way to construct the equivalent DFA from an NFA is to start with the start state and construct new states on the fly as we reach them.

	a	b
→{1}	{1}	{2}
{2}	{2,3}	{3}
{3}	{1,3}	Ø
{2,3}	{1,2,3}	{3}
Ø	Ø	Ø
{1,3}	{1,3}	{2}
{1,2,3}	{1,2,3}	{2,3}

Graphically:



Regular Languages

- Definition: a language is regular if and only if some deterministic finite automaton recognizes it
- Theorem: A language is regular if and only if some nondeterministic finite automaton recognizes it