CS304: Automata and Formal Languages

Lec 6

Nondeterministic Finite Automata (NFA)

Rachit Nimavat

August 8, 2025

Outline

Alphabet Σ: A finite, non-empty set of symbols

String w: A finite sequence of symbols taken from an alphabet

Language L: A set of strings over a particular alphabet

Consider $\Sigma = \{0, 1\}$ and $L = \{0\}^* \{1\} \equiv 0^*1$.

Deterministic Finite Automata (DFA)

Formal Definition

DFA is a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$

'symbol'	description	in our code
Q	finite set of states	$\texttt{state} \in \{0, 1, 2\}$
Σ	underlying finite alphabet	$\Sigma = \{$'0', '1'$\}$
δ	transition function between states	the 'core logic' in C code
$q_0 \in Q$	start state	$\texttt{state} = 0$
$F \subseteq Q$	accepting states	$\texttt{state} = 1$

Deterministic Finite Automata (DFA)

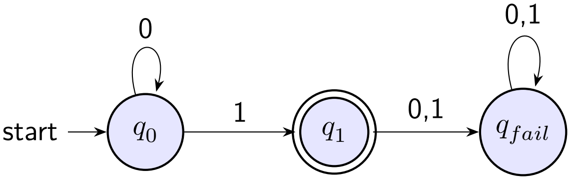
Visualizing DFA

State Diagram is the intuitive way we visualize and work with DFAs

Conventions:

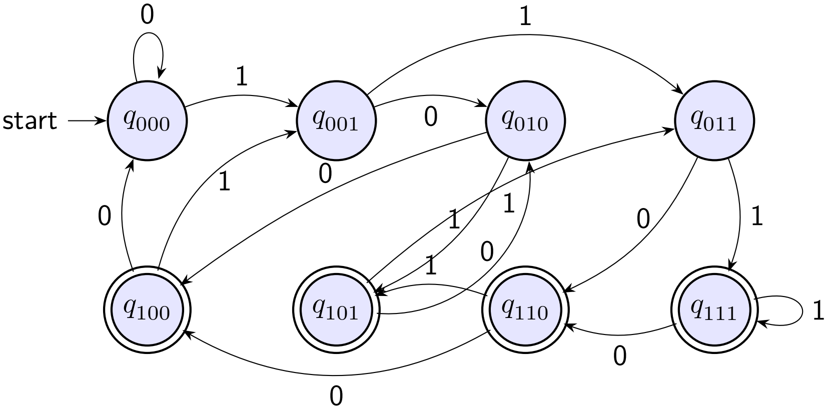
- States Q are circles
- Start state (q_0) has an incoming arrow labeled start
- Accepting states (F) are double circles
- Transitions (δ) are arrows between states, labeled with input symbols from Σ

Our DFA for $L = 0^*1$:



Shortcomings of DFA

Consider $L = \{w \mid$ the third-last symbol in w is $1\}$ over $\Sigma = \{0, 1\}$. A DFA would need to "remember" the last three symbols it has seen.



How can we simplify this mess?

Reason for this complication is:

DFA is very strict!

Pushing the Boundaries: Nondeterminism

DFA is very strict.

What if we relaxed the rules?

- What if a state could have multiple outgoing arrows for the same symbol?
- What if a state could have zero outgoing arrows for a symbol?
- What if a machine could transition between states without reading any input (i.e., on the empty string ε)?

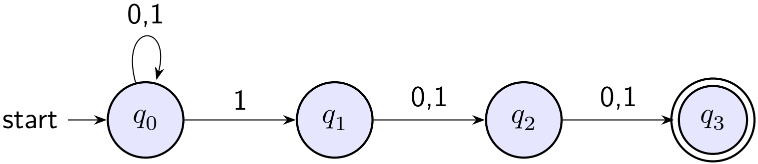
"Nondeterminism"

- Read along until it sees a '1'
- Guess that this '1' might be the third to last symbol
- Verify that exactly two more symbols follow

Nondeterministic Finite Automata : NFA

Consider $L = \{w \mid$ the third-last symbol in w is $1\}$ over $\Sigma = \{0, 1\}$.

- in state q_0, look for 1
- if we see 1, guess we are at third-last position and transition to q_1
- transition to q_2 and q_3 on seeing subsequent symbols
- if we end up at q_3, we accept



much simpler to describe than DFA!

NFA: Formal Definition

An NFA is also a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$, where

- Q: finite set of states
- Σ: finite alphabet
- q_0: single starting state
- F: set of accepting states
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \mapsto 2^Q$
  - $\Sigma \cup \{\varepsilon\}$: input can be a symbol from Σ or the empty string ε
  - 2^Q: also referred as $\mathcal{P}(Q)$ (**power set of** Q), means "the set of all states"

How an NFA Computes: Informal

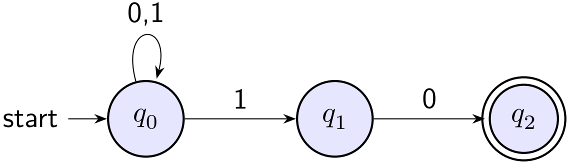
Consider NFA $A = (Q, \Sigma, \delta, q_0, F)$ with input string w.
‘We track its computation as a set of **active states**.

- Initially, “active states” is $\{q_0\}$.
- When you read a symbol x, look at all the states you are currently in. Find all the states you can get to from them by reading x. This new set is your next set of active states.
- Repeat for the whole string w.

Acceptance: Accept w if, after reading all of w, the set of active states contains at least one state of F
Rejection: Reject otherwise, i.e, after reading all of w, the set of active states does not contain any state of F

Example 1

$\Sigma = \{0, 1\}$. $L = \{w \mid w \text{ ends with } 10\}$



Example 2

$\Sigma = \{0, 1\}$. $L = \{w \mid w \text{ contains substring } 11\}$.