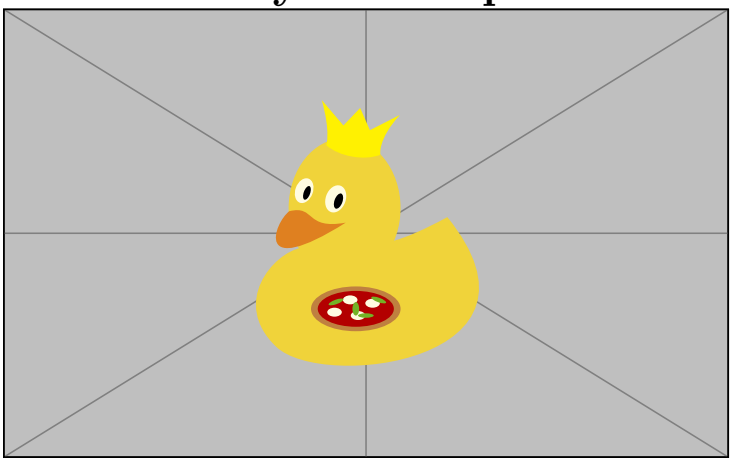
Notes on Deterministic Finite Automata (DFA) A Comprehensive Guide

for Theory of Computation



Generated: October 8, 2025

Contents

1 Introduction to Finite Automata

Finite Automata (FA) are the simplest models of computation. They represent a machine with a finite number of states and are used to recognize patterns in input strings. They have a very limited memory and can only move forward through the input.

There are two main types:

- **Deterministic Finite Automata (DFA):** For any given state and input symbol, there is **exactly one** state to transition to.
- **Nondeterministic Finite Automata (NFA):** For a given state and input symbol, there can be zero, one, or multiple states to transition to.

This document focuses exclusively on DFAs.

2 Formal Definition of a DFA

A Deterministic Finite Automaton (DFA) is formally defined as a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$, where:

1. Q is a **finite set of states**.
2. Σ is a **finite set of input symbols**, called the alphabet.
3. $\delta : Q \times \Sigma \to Q$ is the **transition function**. This function takes the current state and an input symbol and returns the next state.
4. $q_0 \in Q$ is the **start state**.
5. $F \subseteq Q$ is the **set of final (or accepting) states**.

A DFA accepts an input string w if, starting from q_0, the sequence of transitions corresponding to the symbols in w leads to a state in F. The set of all strings that the DFA M accepts is called the **language of the machine**, denoted $\mathcal{L}(M)$.

3 Examples of DFAs

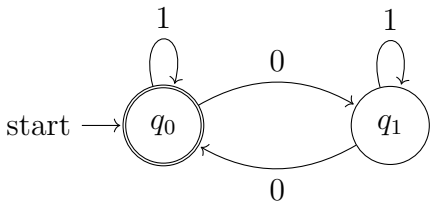
Visual diagrams are the best way to understand DFAs. In these diagrams:

- States (Q) are circles.
- The start state (q_0) has an incoming arrow with no source.
- Final states (F) are double circles.
- Transitions (δ) are arrows between states, labeled with input symbols (Σ).

3.1 Example 1: Language of strings with an even number of 0s

Let $\Sigma = \{0, 1\}$. We want a DFA that accepts strings like ϵ, "11", "00", "1010", etc.

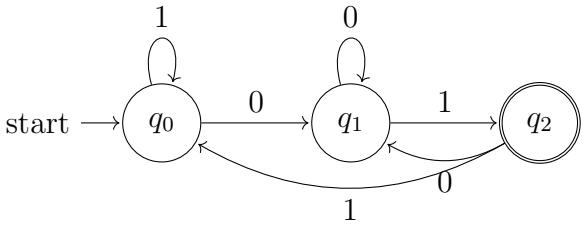
- $Q = \{q_0, q_1\}$
- $\Sigma = \{0, 1\}$
- q_0 is the start state.
- $F = \{q_0\}$
- δ is defined as:
  - $\delta(q_0, 0) = q_1$
  - $\delta(q_0, 1) = q_0$
  - $\delta(q_1, 0) = q_0$
  - $\delta(q_1, 1) = q_1$



3.2 Example 2: Language of strings ending in "01"

Let $\Sigma = \{0, 1\}$. We want a DFA that accepts strings like "01", "001", "1101", etc.

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{0, 1\}$
- q_0 is the start state.
- $F = \{q_2\}$
- δ is defined by the diagram below.



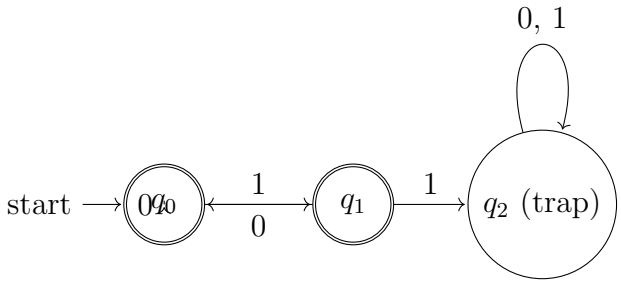
4 Smart Quirks and Properties of DFAs

DFAs have some very elegant and powerful properties that are not always obvious at first glance.

4.1 The Trap State (or Sink State)

A DFA must define a transition for every state and every symbol in Σ. What if a string enters a sequence that guarantees it will be rejected? We use a **trap state**: a non-accepting state that loops back to itself on all inputs. Once a string enters a trap state, it can never leave.

Example: A DFA for strings over $\{0, 1\}$ that do not contain "11".



4.2 Complementation of a DFA

The set of regular languages is **closed under complementation**. This means if a language $\mathcal{L}(M)$ is regular, its complement $\overline{\mathcal{L}(M)}$ (all strings in Σ^* not in $\mathcal{L}(M)$) is also regular.

The Trick: To get a DFA for the complement language, simply flip the final and non-final states. If $M = (Q, \Sigma, \delta, q_0, F)$, then the DFA for the complement is $M' = (Q, \Sigma, \delta, q_0, Q \setminus F)$. Everything that was accepted is now rejected, and everything that was rejected is now accepted.

4.3 Uniqueness of the Minimal DFA

For any regular language, there is a **unique** DFA that accepts it and has the minimum possible number of states (up to isomorphism, i.e., relabeling the states). This is a very powerful theoretical result, as it gives us a canonical representation for any regular language. Algorithms exist to minimize any given DFA to this unique form.

5 Regular Languages and Kleene's Theorem

So, what kinds of languages can DFAs recognize? They can recognize the family of languages known as **Regular Languages**. But this family has another, seemingly different, definition.

A **Regular Expression** is a formal way to specify a pattern of text. For example, $(0|1)^*01$ is a regular expression for the set of strings over $\{0, 1\}$ that end in "01".

This brings us to one of the most fundamental theorems in computer science.

5.1 Kleene's Theorem (KR)

Theorem: A language is regular (i.e., accepted by a DFA) *if and only if* it can be described by a regular expression.

This is a powerful equivalence. It connects two different worlds:

1. The world of **machines** (DFAs, NFAs).

2. The world of **declarative patterns** (regular expressions).

This theorem tells us that any pattern you can write with a regular expression can be recognized by a finite automaton, and vice-versa. The proofs for this are constructive, meaning there are algorithms to convert from one form to the other:

- **Regular Expression → NFA:** Thompson's Construction Algorithm is a famous method for this. Since every NFA can be converted to an equivalent DFA, this proves one direction.
- **DFA → Regular Expression:** State Elimination is a common algorithm. It systematically removes states from the DFA and labels the transitions with increasingly complex regular expressions until only the start and final states remain.