Automata Based Programming Paradigm

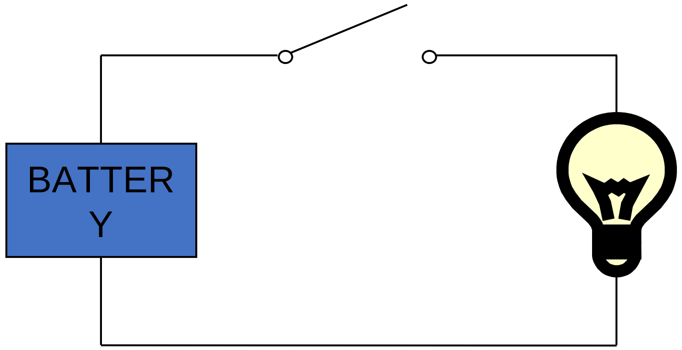
Introduction

Automata-based programming is a programming paradigm in which the program or its part is thought of as a model of a finite state machine or any other formal automation.

What is Automata Theory?

- Automata theory is the study of abstract computational devices
- Abstract devices are (simplified) models of real computations
- Computations happen everywhere: On your laptop, on your cell phone, in nature, …

Example:



input: switch

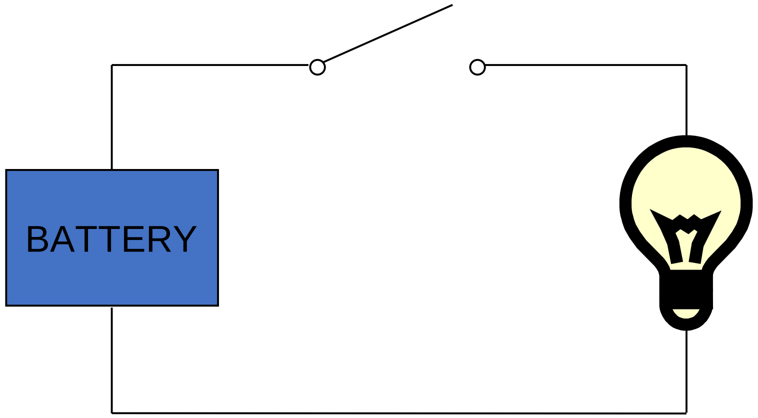
output: light bulb

actions: flip switch

states: on, off

Simple Computer

Example:

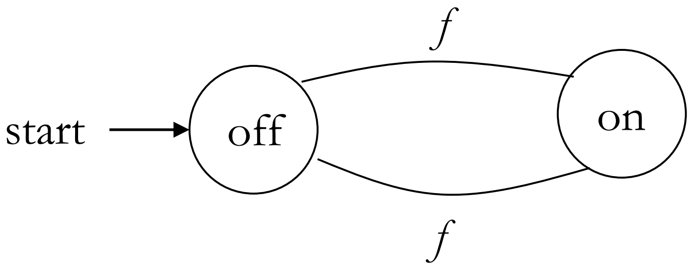


input: switch

output: light bulb

actions: flip switch

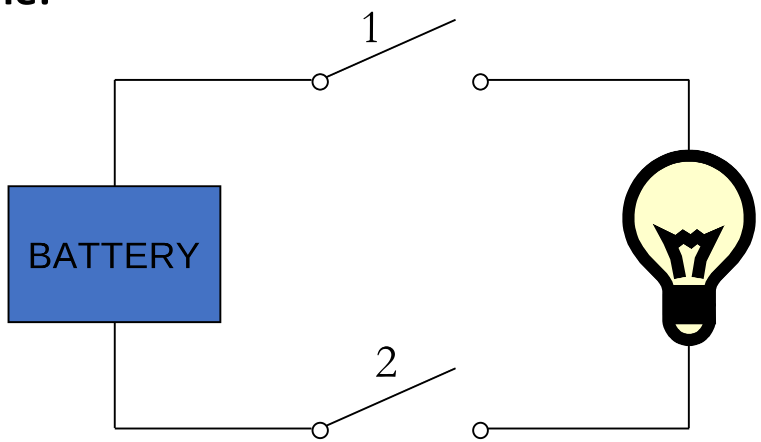
states: on, off



bulb is on if and only if there was an odd number of flips

Another “computer”

Example:

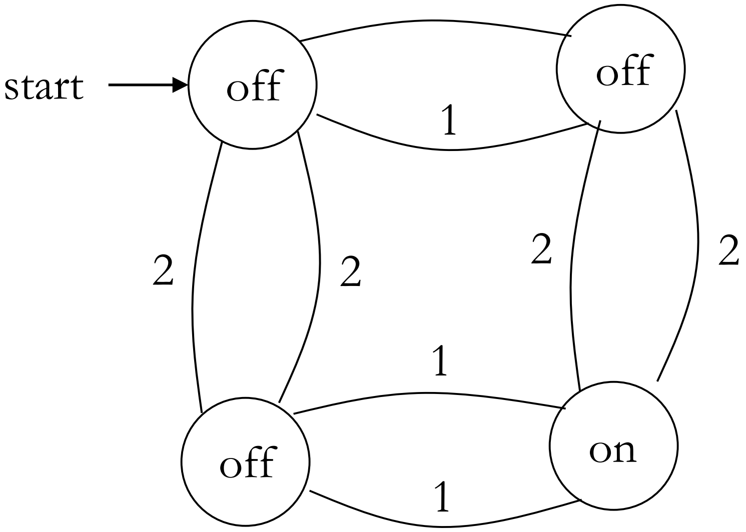


inputs: switches 1 and 2

actions: 1 for “flip switch 1”

actions: 2 for “flip switch 2”

states: on, off



bulb is on if and only if both switches were flipped an odd number of times

Types of Automata

finite automata	Devices with a finite amount of memory. Used to model “small” computers.
push-down automata	Devices with infinite memory that can be accessed in a restricted way. Used to model parsers, etc.
Turing Machines	Devices with infinite memory. Used to model any computer.

Alphabets and strings

A common way to talk about words, number, pairs of words, etc. is by representing them as strings

To define strings, we start with an alphabet

An alphabet is a finite set of symbols.

Examples:

$\Sigma_1 = \{a, b, c, d, \ldots, z\}$: the set of letters in English

$\Sigma_2 = \{0, 1, \ldots, 9\}$: the set of (base 10) digits

$\Sigma_3 = \{a, b, \ldots, z, \#\}$: the set of letters plus the special symbol #

$\Sigma_4 = \{ (,) \}$: the set of open and closed brackets

Strings

A **string** over alphabet Σ is a finite sequence of symbols in Σ.

The empty string will be denoted by e

Examples:

abfbz is a string over $\Sigma_1 = \{a, b, c, d, \ldots, z\}$

9021 is a string over $\Sigma_2 = \{0, 1, \ldots, 9\}$

ab#bc is a string over $\Sigma_3 = \{a, b, \ldots, z, \#\}$

))()(() is a string over $\Sigma_4 = \{ (,) \}$

Languages

A language is a set of strings over an alphabet.

Languages can be used to describe problems with “yes/no” answers, for example:

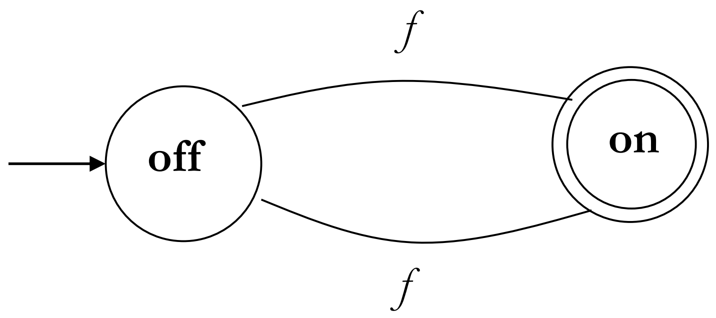
$L_1 =$ The set of all strings over Σ_1 that contain the substring “SRM”

$L_2 =$ The set of all strings over Σ_2 that are divisible by 7 $= \{7, 14, 21, \ldots\}$

$L_3 =$ The set of all strings of the form $\mathrm{s\#s}$ where s is any string over $\{\mathrm{a, b, \ldots, z}\}$

$L_4 =$ The set of all strings over Σ_4 where every (can be matched with a subsequent)

Finite Automata



There are states off and on, the automaton starts in off and tries to reach the “good state” on

What sequences of fs lead to the good state?

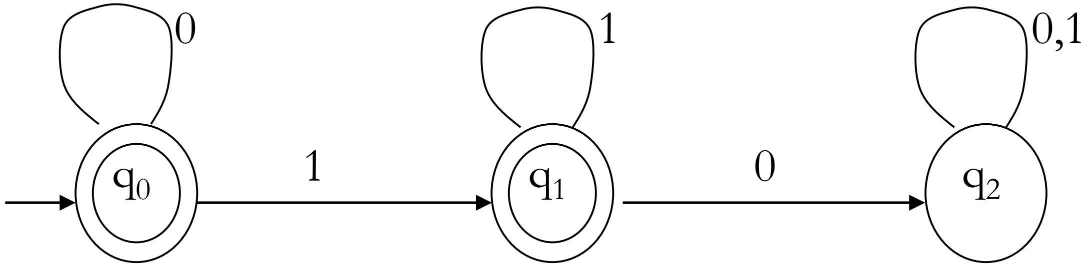
Answer: {f, fff, fffff, …} = {f n: n is odd}

This is an example of a deterministic finite automaton over alphabet {f}

Deterministic finite automata

- A deterministic finite automaton (DFA) is a 5-tuple $(\mathcal{Q}, \Sigma, \delta, \mathrm{q}_0, F)$ where
  - $\mathcal{Q}$ is a finite set of states
  - Σ is an alphabet
  - $\delta: \mathcal{Q} \times \Sigma \to \mathcal{Q}$ is a transition function
  - $\mathrm{q}_0 \in \mathcal{Q}$ is the initial state
  - $F \subseteq \mathcal{Q}$ is a set of accepting states (or final states).
- In diagrams, the accepting states will be denoted by double loops

Example



alphabet $\Sigma = \{0, 1\}$
start state $\mathcal{Q} = \{q_0, q_1, q_2\}$
initial state q_0
accepting states $F = \{q_0, q_1\}$

transition function δ:

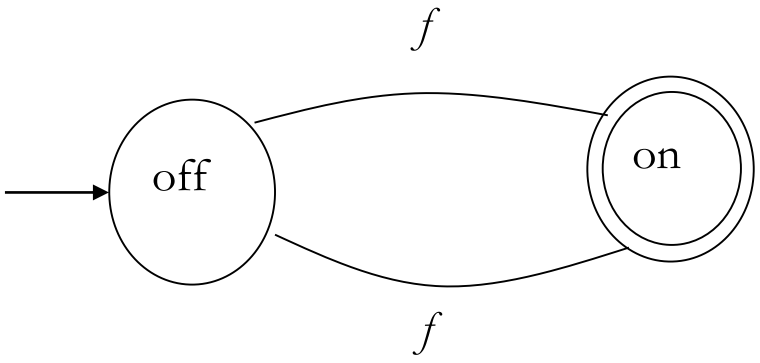
inputs

states	0	1
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_2	q_2

Language of a DFA

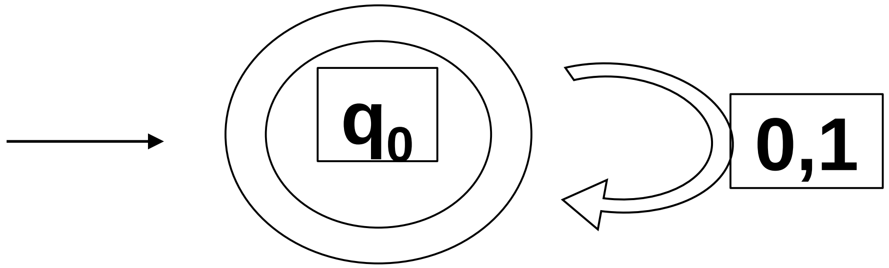
The language of a DFA $(Q, \Sigma, \delta, q_0, F)$ is the set of all strings over Σ that, starting from q_0 and following the transitions as the string is read left to right, will reach some accepting state.

M:

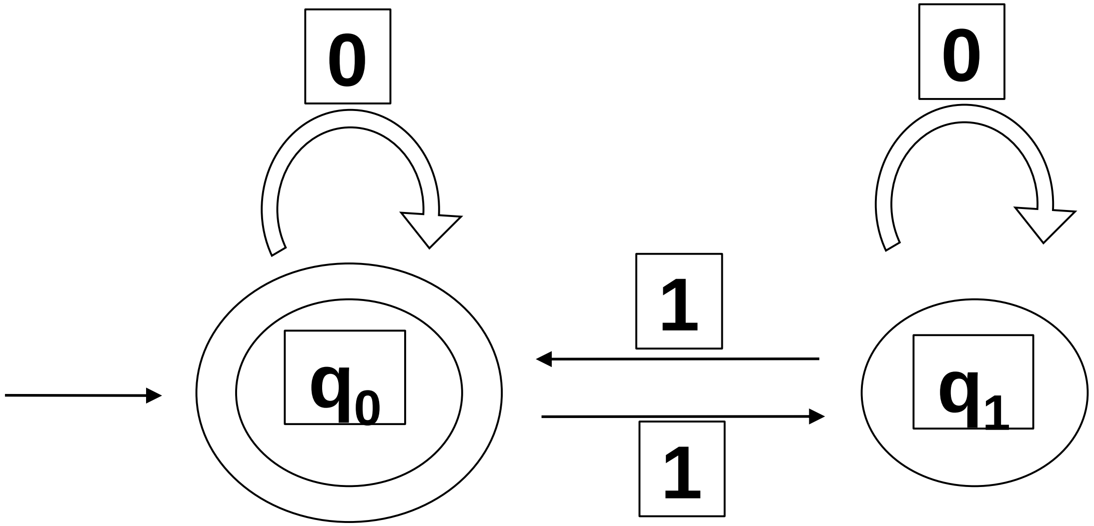


- Language of M is $\{f, fff, fffff, \ldots\} = \{f^n: n \text{ is odd}\}$

Example of DFA



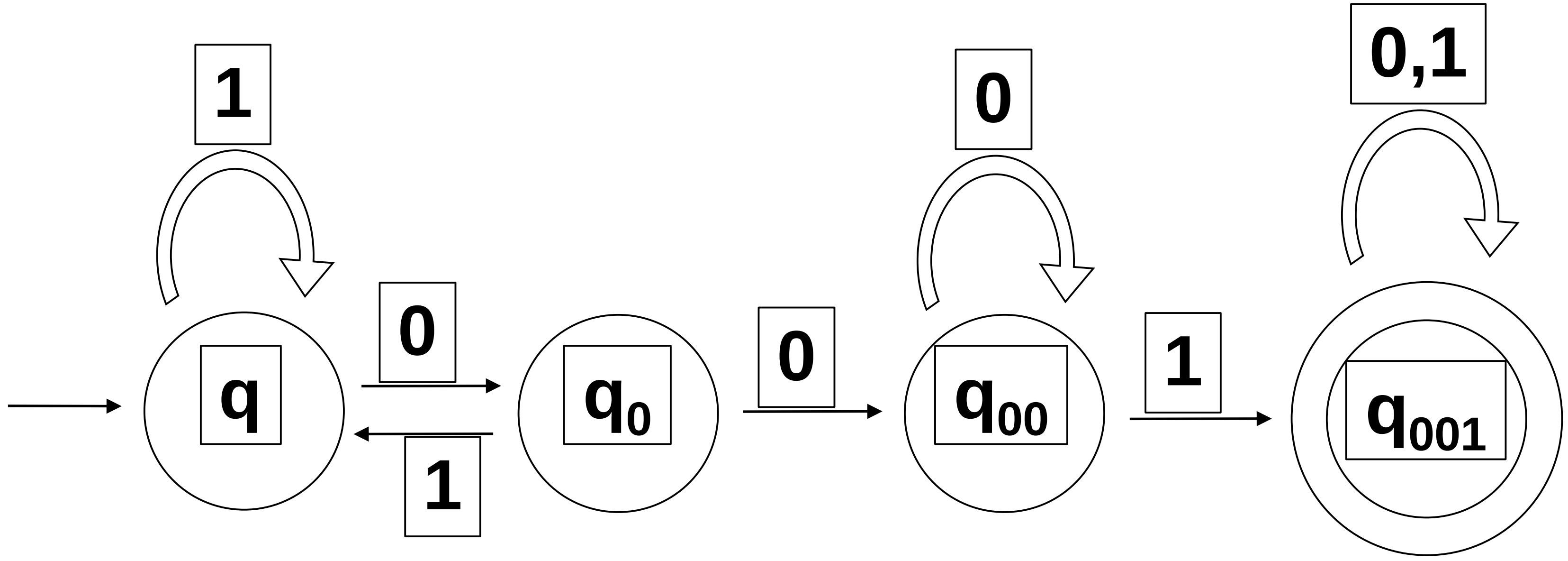
L(M) = {0,1}*



L(M) = { w | w has an even number of 1s}

Example of DFA

Build an automaton that accepts all and only those strings that contain 001



Example of DFA using Python

```
from automata.fa.dfa import DFA
# DFA which matches all binary strings ending in an odd number of '1's
dfa = DFA(
    states={'q0', 'q1', 'q2'},
    input_symbols={'0', '1'},
    transitions={
        'q0': {'0': 'q0', '1': 'q1'},
        'q1': {'0': 'q0', '1': 'q2'},
        'q2': {'0': 'q2', '1': 'q1'}
    },
    initial_state='q0',
    final_states={'q1'}
)
dfa.read_input('01')   # answer is  'q1'
dfa.read_input('011')  # answer is error
print(dfa.read_input_stepwise('011'))
Answer # yields:
# 'q0'     # 'q0'     # 'q1'
# 'q2'     # 'q1'
```

```
if dfa.accepts_input('011'):
    print('accepted')
else:
    print('rejected')
```

Questions for DFA

c) A DFA that accepts all strings that contain 010 or do not contain 0.

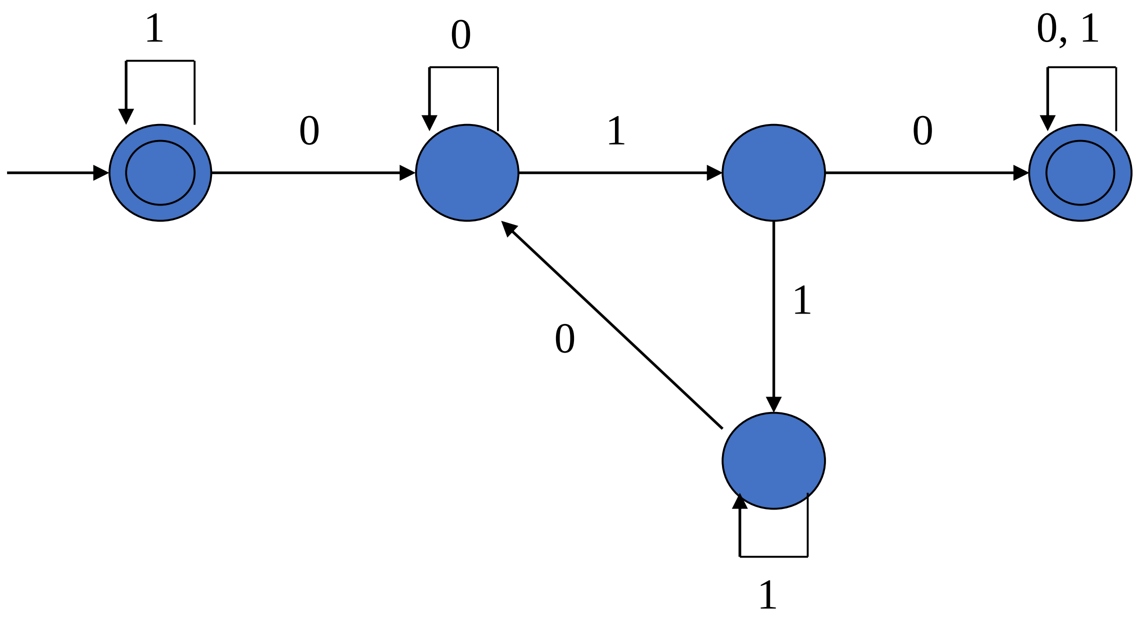
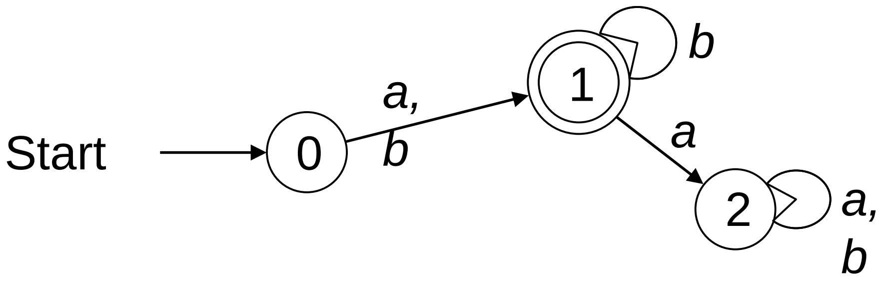


Table Representation of a DFA

A DFA over A can be represented by a transition function T : States X A -> States, where T(i, a) is the state reached from state i along the edge labelled a, and we mark the start and final states. For example, the following figures show a DFA and its transition table.



	T	a	b
start	0	1	1
final	1	2	1
	2	2	2

Sample Exercises - DFA

1. Write a automata code for the Language that accepts all and only those strings that contain 001
2. Write a automata code for L(M) ={ w | w has an even number of 1s}
3. Write a automata code for L(M) ={0,1}*
4. Write a automata code for L(M)=$a + aa^*b$.
5. Write a automata code for L(M)=$\{(ab)^n \mid n \in \mathrm{N}\}$
6. Write a automata code for Let $\Sigma = \{0, 1\}$.

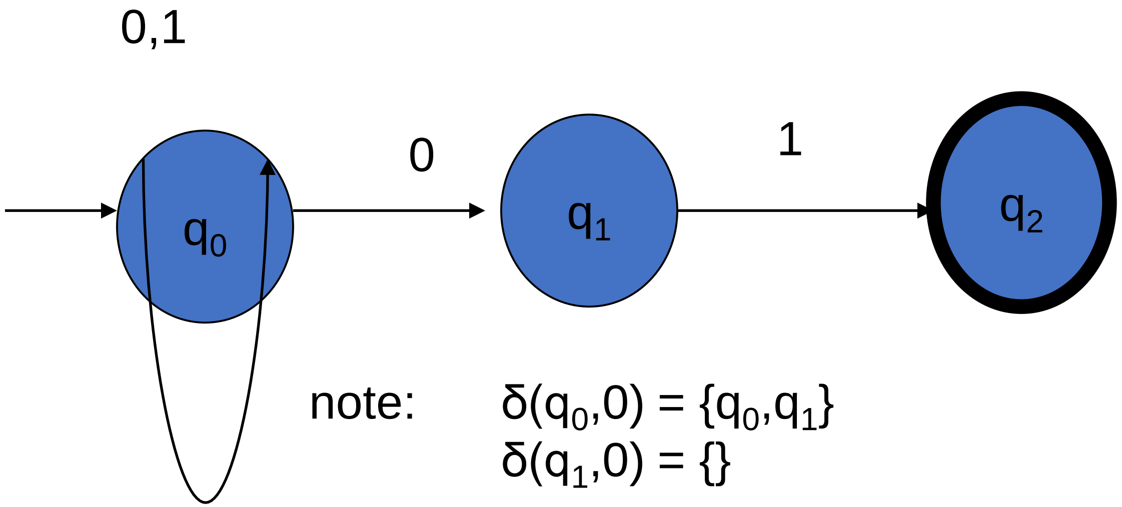
   Given DFAs for {}, {ε}, Σ^*, and Σ^+.

NDFA

- A nondeterministic finite automaton M is a five-tuple $M = (Q, \Sigma, \delta, q_0, F)$, where:
  - Q is a finite set of states of M
  - Σ is the finite input alphabet of M
  - $\delta: Q \times \Sigma \rightarrow$ power set of Q, is the state transition function mapping a state-symbol pair to a subset of Q
  - q_0 is the start state of M
  - $F \subseteq Q$ is the set of accepting states or final states of M

Example NDFA

- NFA that recognizes the language of strings that end in 01



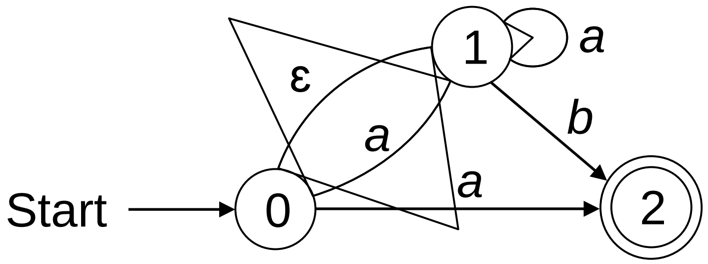
note: $\delta(q_0,0) = \{q_0,q_1\}$
$\delta(q_1,0) = \{\}$

Exercise: Draw the complete transition table for this NFA

NDFA

A nondeterministic finite automaton (NFA) over an alphabet A is similar to a DFA except that epislon-edges are allowed, there is no requirement to emit edges from a state, and multiple edges with the same letter can be emitted from a state.

Example. The following NFA recognizes the language of a + aa*b + a*b.



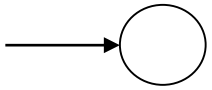
	T	a	b	e
start	0	{1, 2}	$\varnothing$	{1}
	1	{1}	{2}	$\varnothing$
final	2	$\varnothing$	$\varnothing$	$\varnothing$

Table representation of NFA

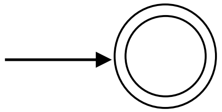
An NFA over A can be represented by a function T : States × A ∪ {L} → power(States), where T(i, a) is the set of states reached from state i along the edge labeled a, and we mark the start and final states. The following figure shows the table for the preceding NFA.

Examples

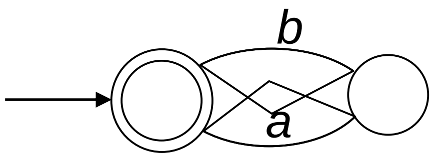
Solutions: (a): Start



(b): Start



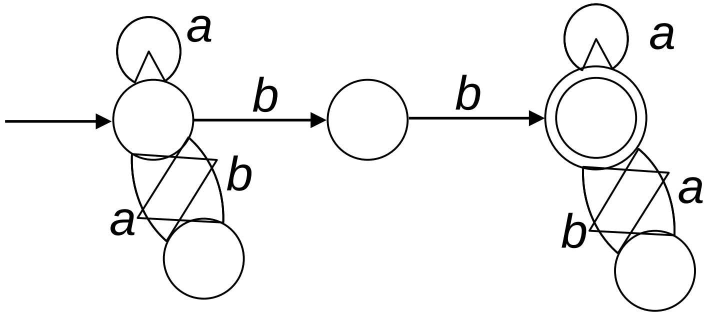
(c): Start



Find an NFA to recognize the language (a + ba)*bb(a + ab)*.

A solution:

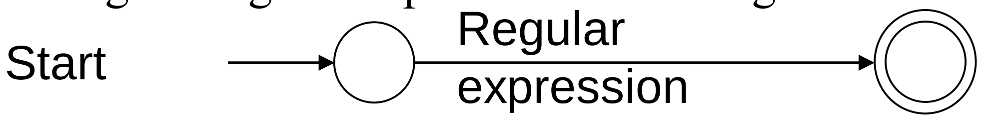
Start



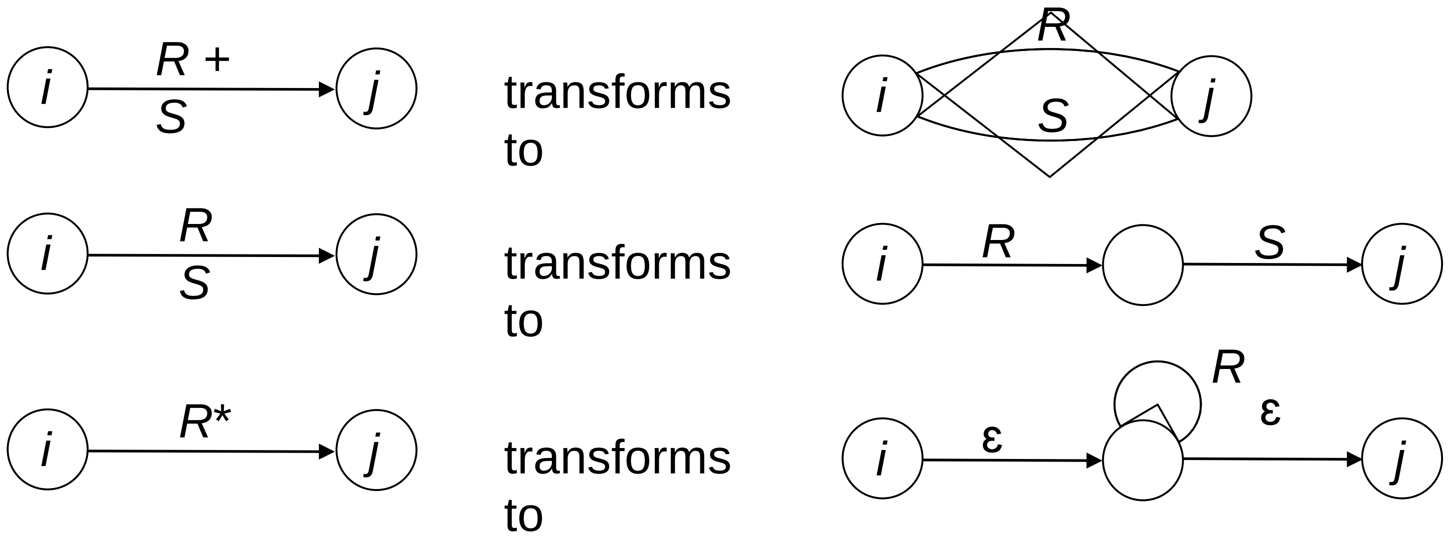
Examples

Algorithm: *Transform a Regular Expression into a Finite Automaton*

Start by placing the regular expression on the edge between a start and final state:

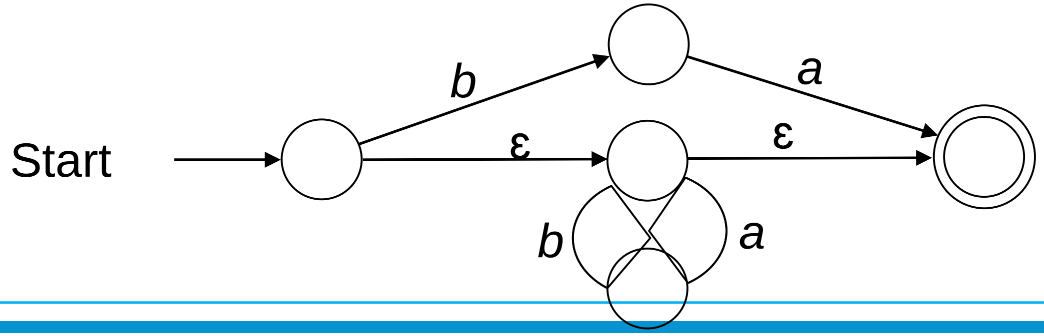


Apply the following rules to obtain a finite automaton after erasing any ∅-edges.



Quiz. Use the algorithm to construct a finite automaton for (ab)* + ba.

***Answer*:**



Example of NFA using Python

```
from automata.fa.nfa import NFA
# NFA which matches strings beginning with 'a', ending with 'a', and
containing
# no consecutive 'b's
nfa = NFA(
    states={'q0', 'q1', 'q2'},
    input_symbols={'a', 'b'},
    transitions={
        'q0': {'a': {'q1'}},
        # Use '' as the key name for empty string (lambda/epsilon)
transitions
        'q1': {'a': {'q1'}, '': {'q2'}},
        'q2': {'b': {'q0'}}
    },
    initial_state='q0',
    final_states={'q1'}
)
```

```
nfa.read_input('aba')
ANSWER :{'q1', 'q2'}

nfa.read_input('abba')
ANSWER: ERROR


nfa.read_input_stepwise('aba')

if nfa.accepts_input('aba'):
    print('accepted')
else:
    print('rejected')
ANSWER: ACCEPTED
nfa.validate()
ANSWR: TRUE
```

Sample Exercises - NFA

1. Write a automata code for the Language that accepts all end with 01
2. Write a automata code for L(M)= $a + aa^*b + a^*b$.
3. Write a automata code for Let $\Sigma = \{0, 1\}$.

   Given NFAs for {}, {ε}, $\{(ab)^n \mid n \in N\}$, which has regular expression $(ab)^*$.