Module I

Introduction to Automata Theory and its significance. Type 3 Formalism: Finite state automata – Properties of transition functions, Designing finite automata, NFA, Finite Automata with Epsilon Transitions, Equivalence of NFA and DFA, Conversion of NFA to DFA, Equivalence and Conversion of NFA with and without Epsilon Transitions.

Module II

Myhill-Nerode Theorem, Minimal State FA Computation. Finite State Machines with Output- Mealy and Moore machine (Design Only), Two- Way Finite Automata. Regular Grammar, Regular Expressions, Equivalence of regular expressions and NFA with epsilon transitions. Converting Regular

Expressions to NFA with epsilon transitions Equivalence of DFA and regular expressions, converting DFA to Regular Expressions.

Text Books

John E Hopcroft, Rajeev Motwani and Jeffrey D Ullman, ***Introduction to Automata Theory, Languages, and Computation***, 3/e, Pearson Education, 2007

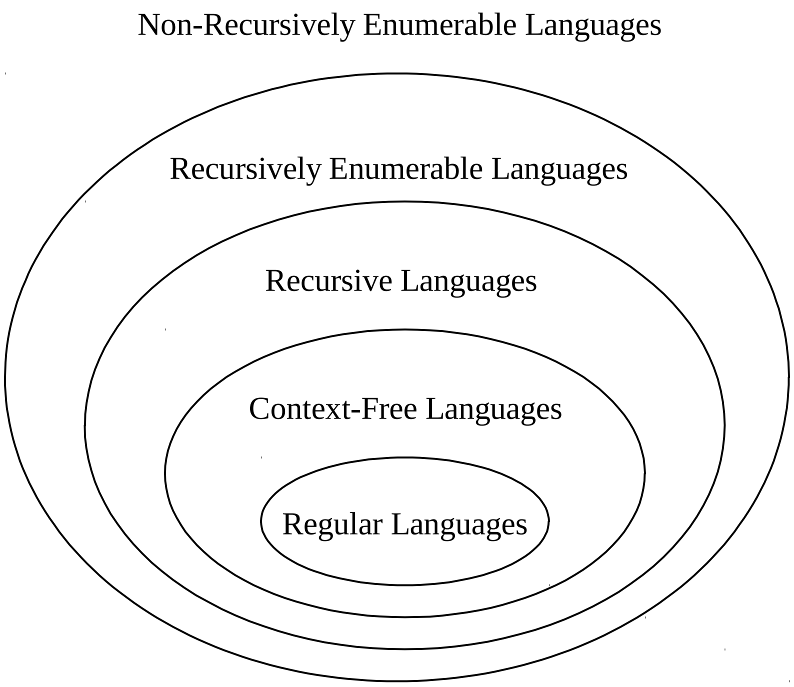
John C Martin, ***Introduction to Languages and the Theory of Computation***, TMH, 2007

Michael Sipser, ***Introduction To Theory of Computation***, Cengage Publishers, 2013

CO1 : Classify formal languages into regular, context- free, context sensitive and unrestricted languages.

CO2 : Design finite state automata, regular grammar, regular expression and Myhill-Nerode relation representations for regular languages.

Hierarchy of languages

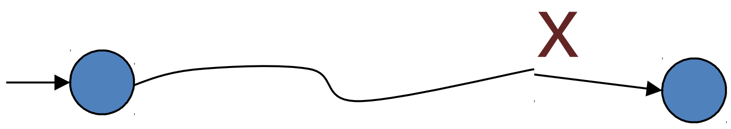


Language	Machine	Grammar
Regular	**Finite Automaton**	Regular Expression, Regular Grammar
Context-Free	Pushdown Automaton	Context-Free Grammar
Recursively Enumerable	Turing Machine	Unrestricted Phrase-Structure Grammar

A man with a wolf, goat and cabbage is on left bank of a river. There is a boat large enough to carry man and only one of the other three. The man and his entourage wish to cross to the right bank, and the man can ferry each across, one at a time. However , if the man leaves wolf and goat unattended on either shore, the wolf will surely eat the goat. Similarly if the cabbage and goat left unattended , the goat will eat cabbage. Is it possible to cross the river without the goat or cabbage being eaten?

Finite Automata

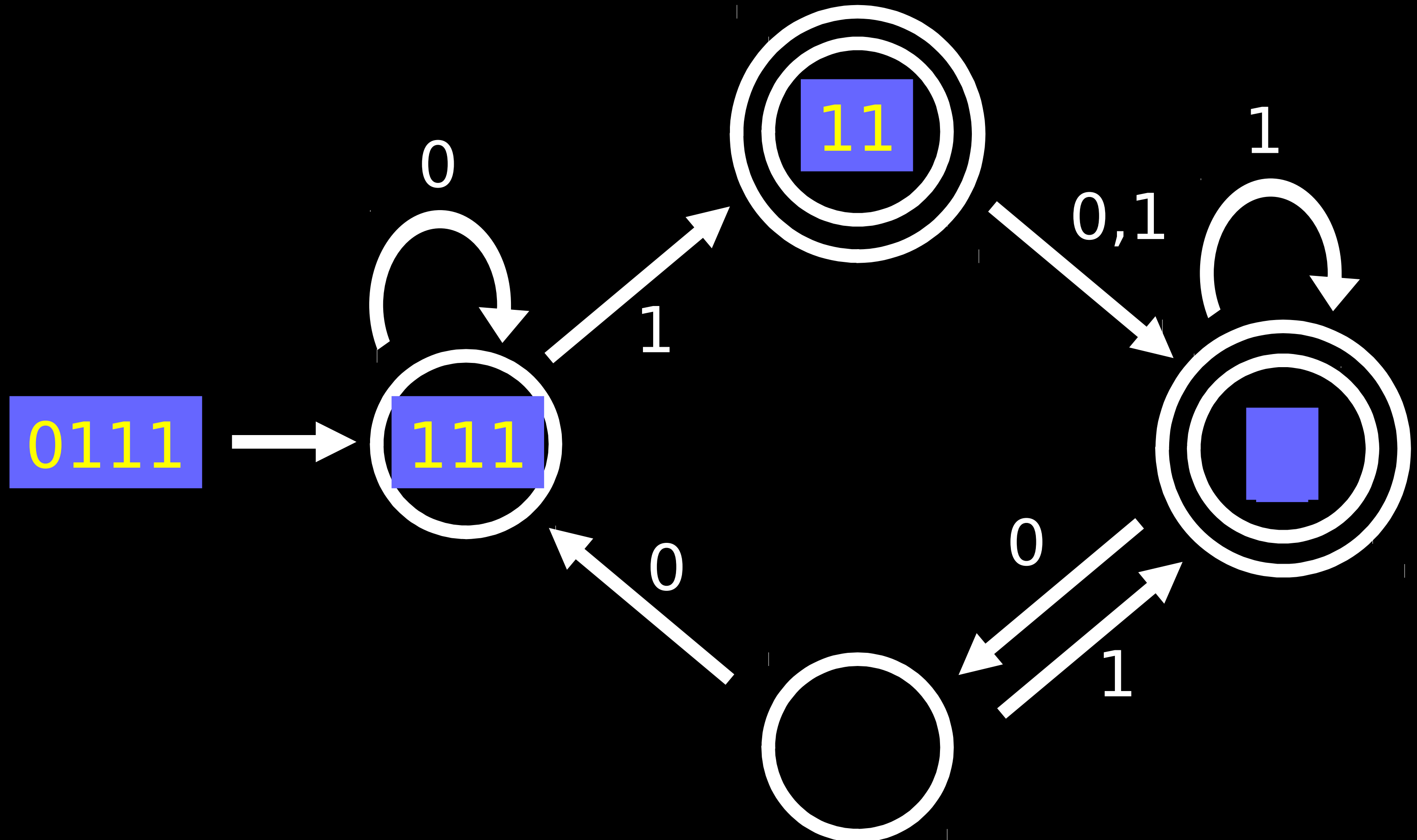
Deterministic Finite Automata

- **Determinism**: Always traverses the *same path* and yields the *same result* for the *same input*
- For every string x, there is a ***unique*** path from initial state and associated with x.
- 

- x is accepted if and only if this path ends at an accept state.

Deterministic Finite Automata

- A Finite State Machine consists of:
  - A finite set of *states*
    - Exactly one “start state”
    - One or more final (“accepting”) states
  - A set of input *alphabet*
  - A transition *function*



The machine accepts a string if the process ends in a double circle

DFA

Mathematical Definition

- A Quintuple:

- 1) A set of *states*, **Q**
- 2) A set of *input alphabet*, **Σ**
- 3) A *transition function*, $\boldsymbol{\delta: Q \times \Sigma \to Q}$
- 4) An initial state, $\mathbf{q_0}$
- 5) A set of final states, $\mathbf{F \subseteq Q}$

DFA Example

- $\mathbf{M} = (\{\mathbf{q_0},\mathbf{q_1},\mathbf{q_2}\}, \{\mathbf{0},\mathbf{1}\}, \boldsymbol{\delta}, \mathbf{q_0}, \{\mathbf{q_1}\})$

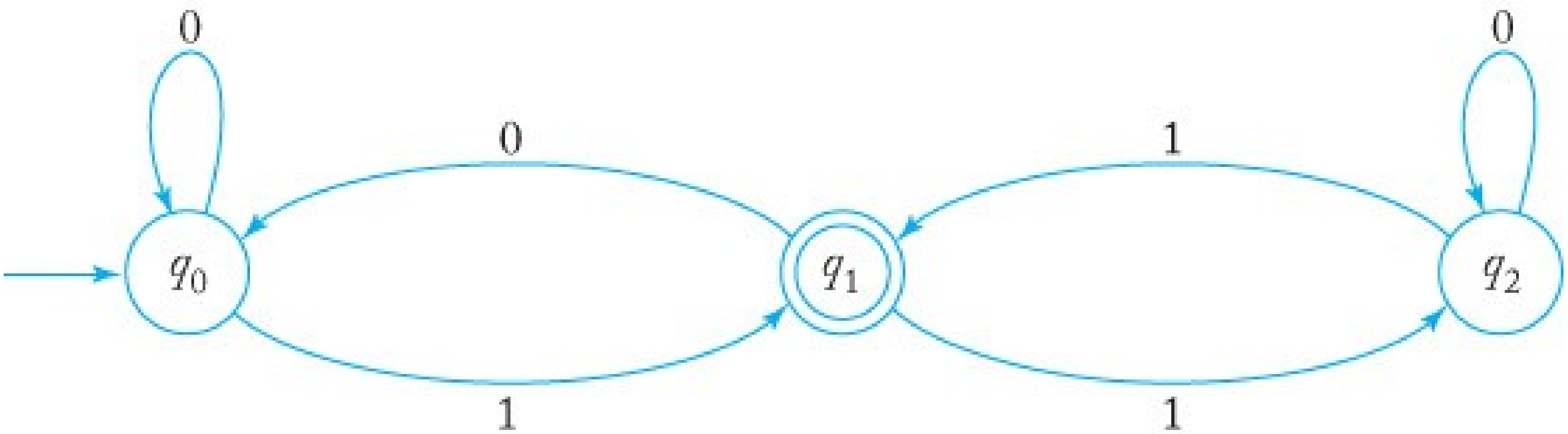
- $\boldsymbol{\delta}$ is defined as:

$$\boldsymbol{\delta}(\mathbf{q_0},\mathbf{0}) = \mathbf{q_0} \qquad \boldsymbol{\delta}(\mathbf{q_0},\mathbf{1}) = \mathbf{q_1}$$
$$\boldsymbol{\delta}(\mathbf{q_1},\mathbf{0}) = \mathbf{q_0} \qquad \boldsymbol{\delta}(\mathbf{q_1},\mathbf{1}) = \mathbf{q_2}$$
$$\boldsymbol{\delta}(\mathbf{q_2},\mathbf{0}) = \mathbf{q_2} \qquad \boldsymbol{\delta}(\mathbf{q_2},\mathbf{1}) = \mathbf{q_1}$$

Transition Table for M

	0	1
$-q_0$	q_0	q_1
$+q_1$	q_0	q_2
q_2	q_2	q_1

Transition Graph for M

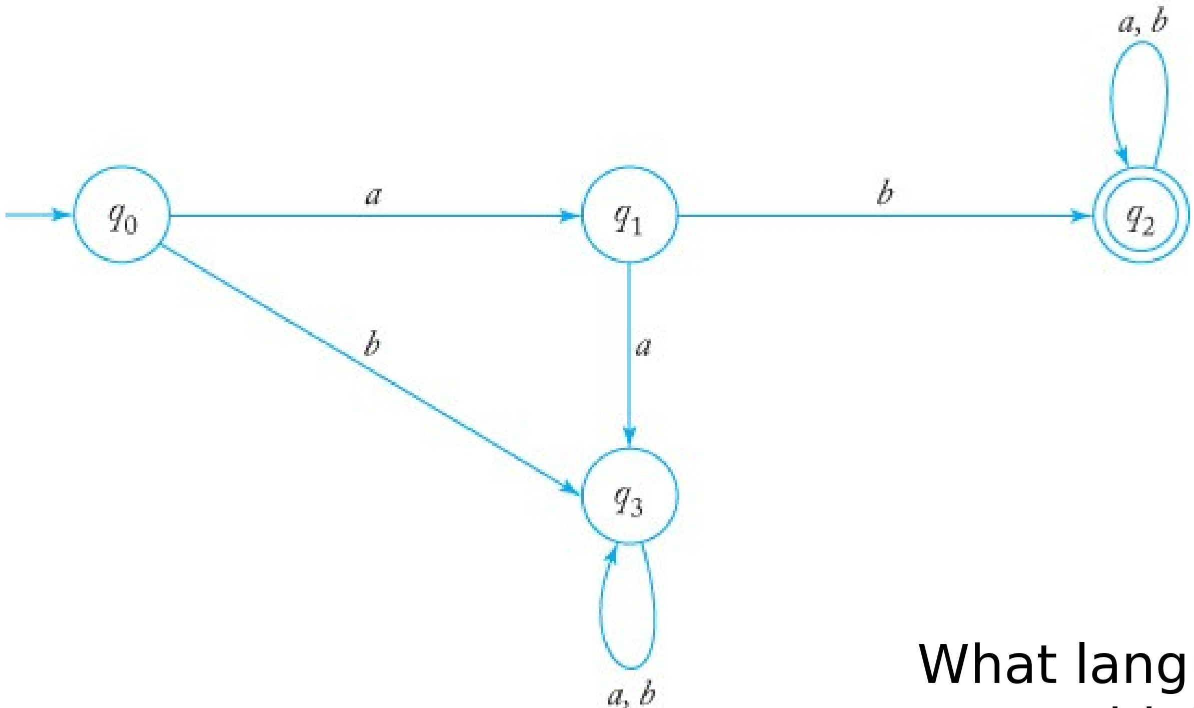


- Each state has exactly 2 out-edges (1 for each letter)

What language is this?

Trap States

- Sometimes a state can never be exited
  - A “black hole” :-)



What language is this?

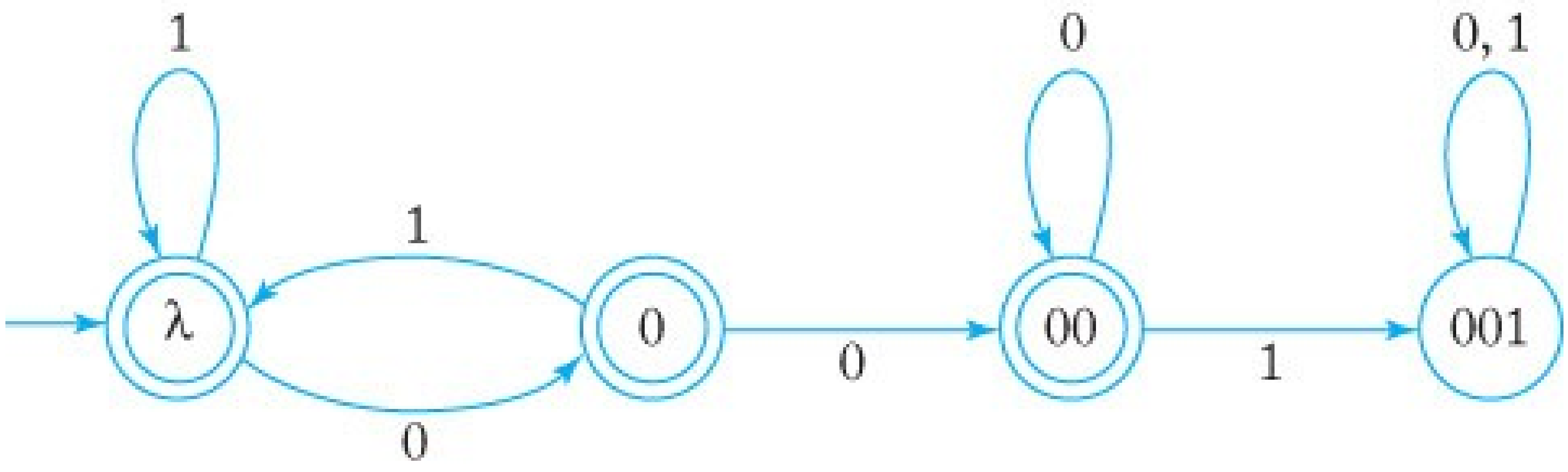
The Complement of a Language

- If we have a DFA for a language, **L**, how can we form a DFA for its *complement*?
  - $\mathbf{\Sigma^* - L}$
- Think of the roles of final states in recognizing strings…

Complement Example

- Find a DFA for the set of all strings *except* those that contain “001” as a substring
- First build a DFA that accepts strings containing “001”
- Then *invert* the “acceptability” of each state
  - Now all other strings will be accepted

Solution



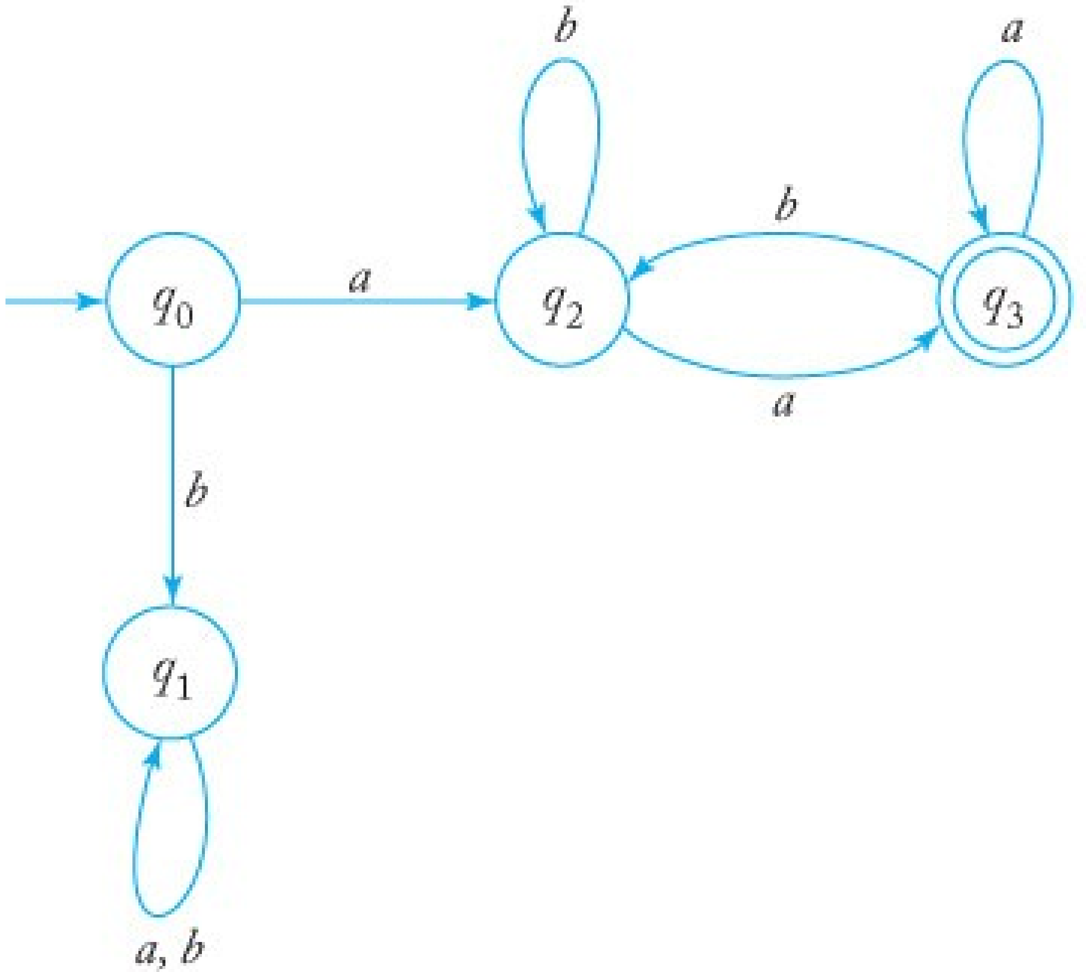
Note that the empty string (**λ**) is accepted

More Practice

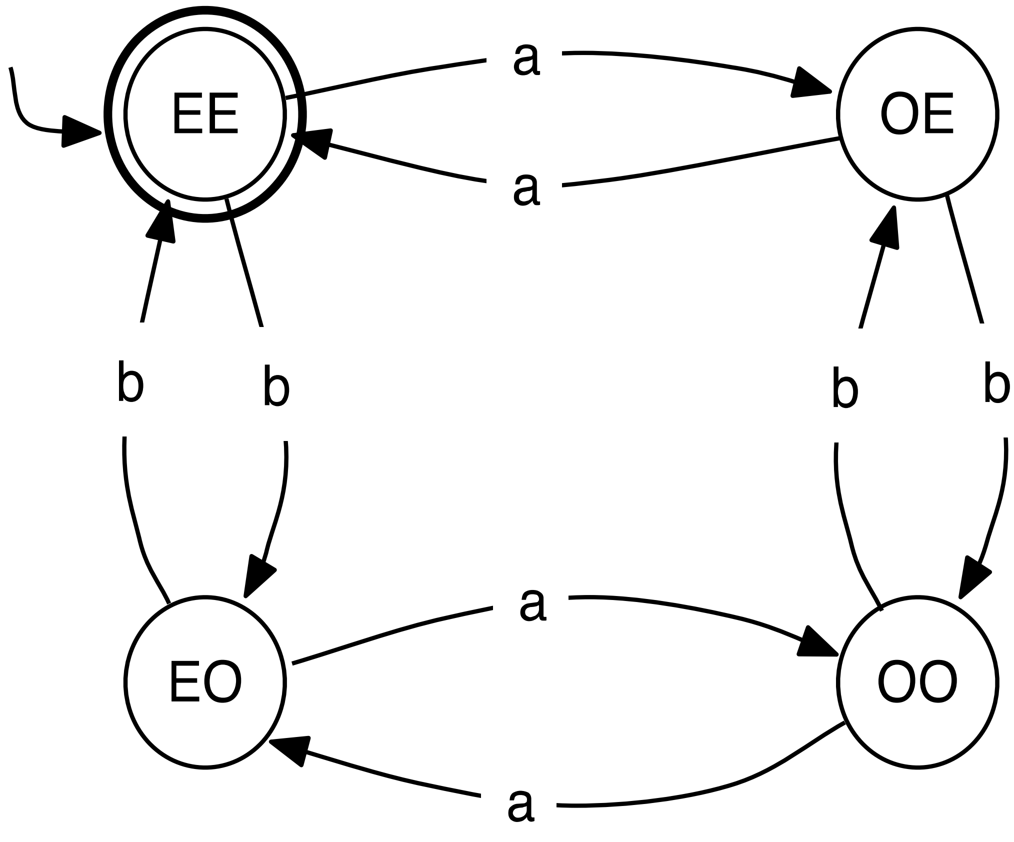
- Build a DFA that accepts all strings over {**a**, **b**} that *start* and *end* in the letter **a** (but not just **"a"**)

- Build a DFA for the language over {**a**, **b**} containing an *even number* of *both* **a**'s and **b**'s

DFA for L



DFA for EVEN-EVEN

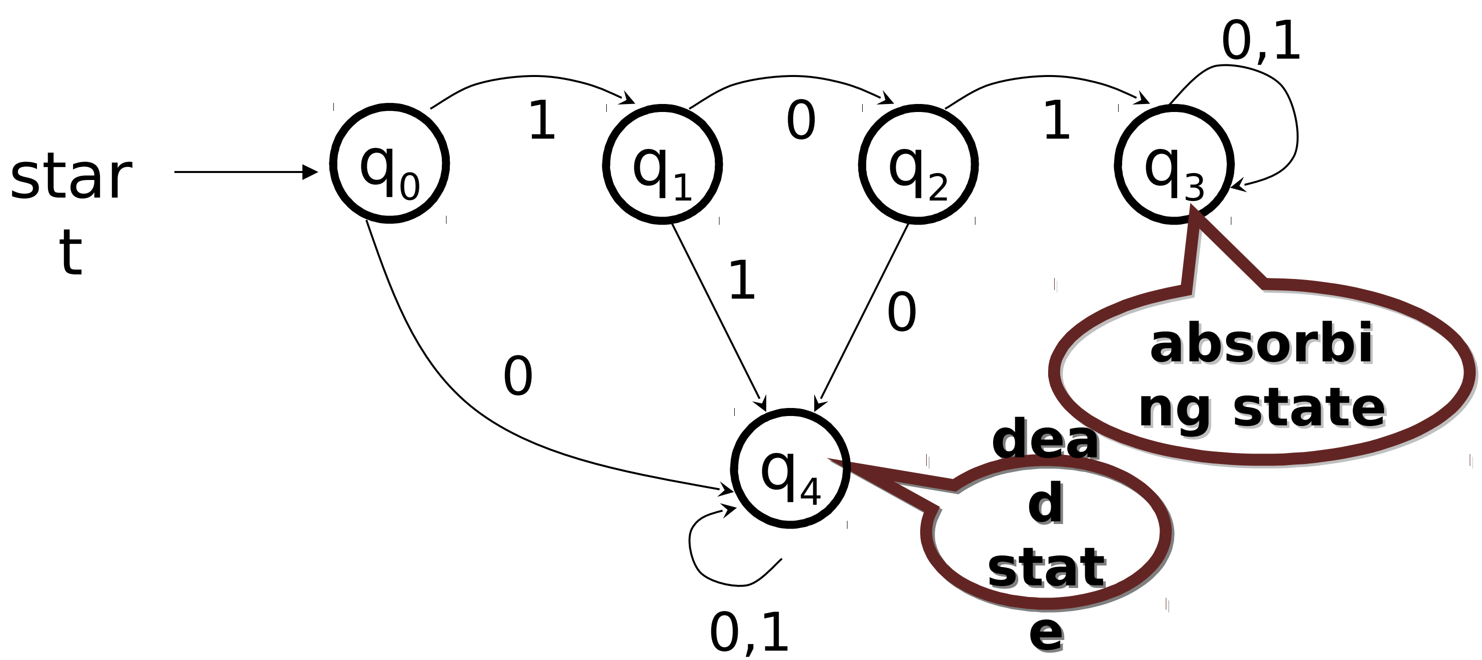


Even More Practice

- Consider *binary numbers* as input strings:
  - Construct a DFA where each state represents the remainder of the number **mod 3**
    - Need 3 states representing 0, 1 and 2, respectively
    - Making the 0–state final accepts numbers ≡ 0 mod 3
    - Making the 1–state final accepts numbers ≡ 1 mod 3
    - Making the 2–state final accepts numbers ≡ 2 mod 3

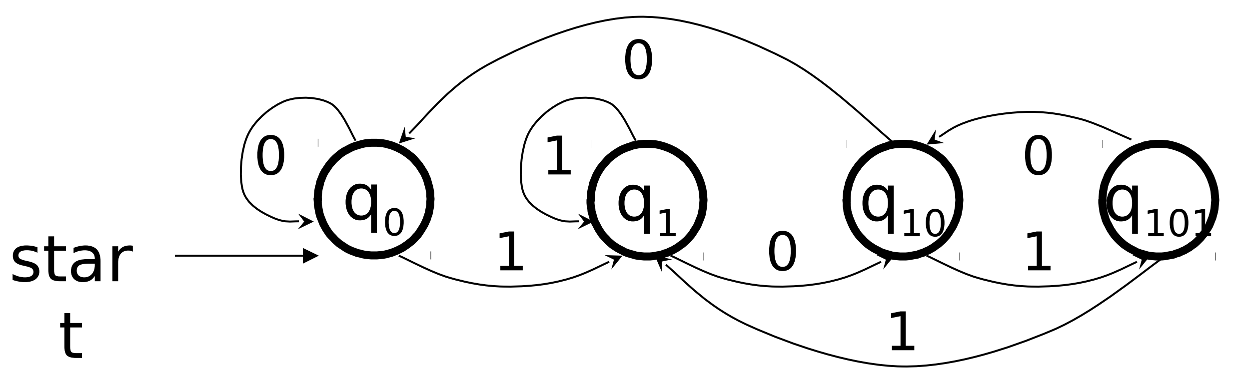
Strings With Common Prefix

- Construct a DFA that accepts a language L over Σ = {0, 1} such that L is the set of all strings starting with “101”.



Strings With Common Suffix

- Construct a DFA that accepts a language L over Σ = {0, 1} such that L is the set of all strings ***ending*** with “101”.



NFA

- For any string x, there may exist ***none*** or ***more than one*** path from initial state and associated with x.

- x is accepted if there is **some** path that ends at an accept state.

NFA

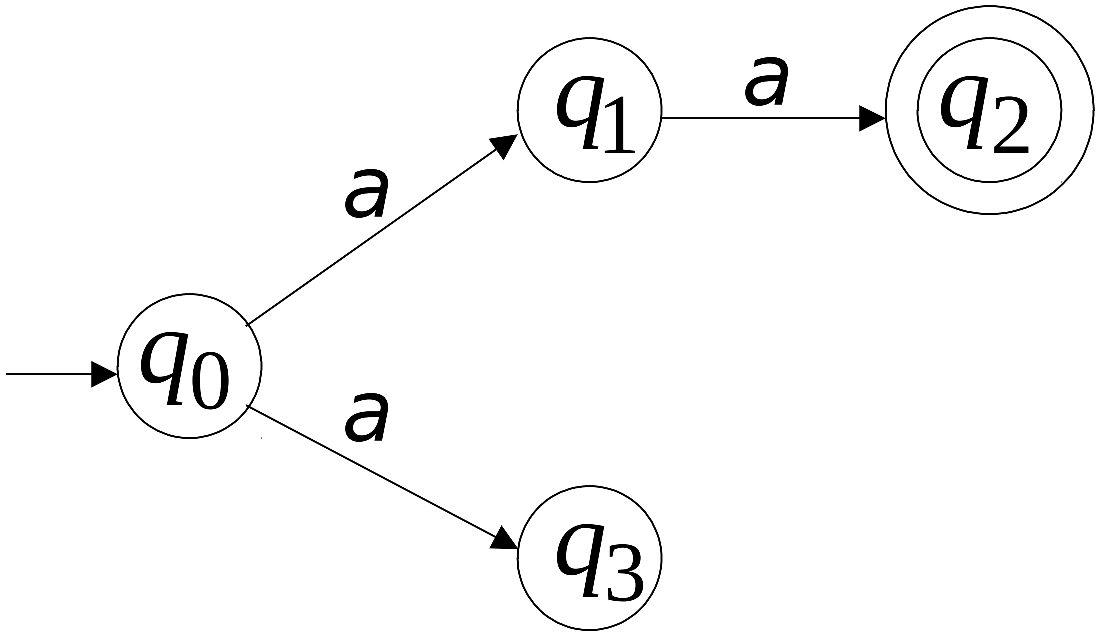
Mathematical Definition

- A Quintuple:
  1) A set of *states*, **Q**
  2) A set of *input alphabet*, **Σ**
  3) A *transition function*, $\boldsymbol{\delta: Q \times \Sigma \rightarrow 2^Q}$
  4) An initial state, $\mathbf{q_0}$
  5) A set of final states, $\mathbf{F \subseteq Q}$

- **Note**: DFAs are a *special case* of NFAs

Nondeterministic Finite Accepter (NFA)

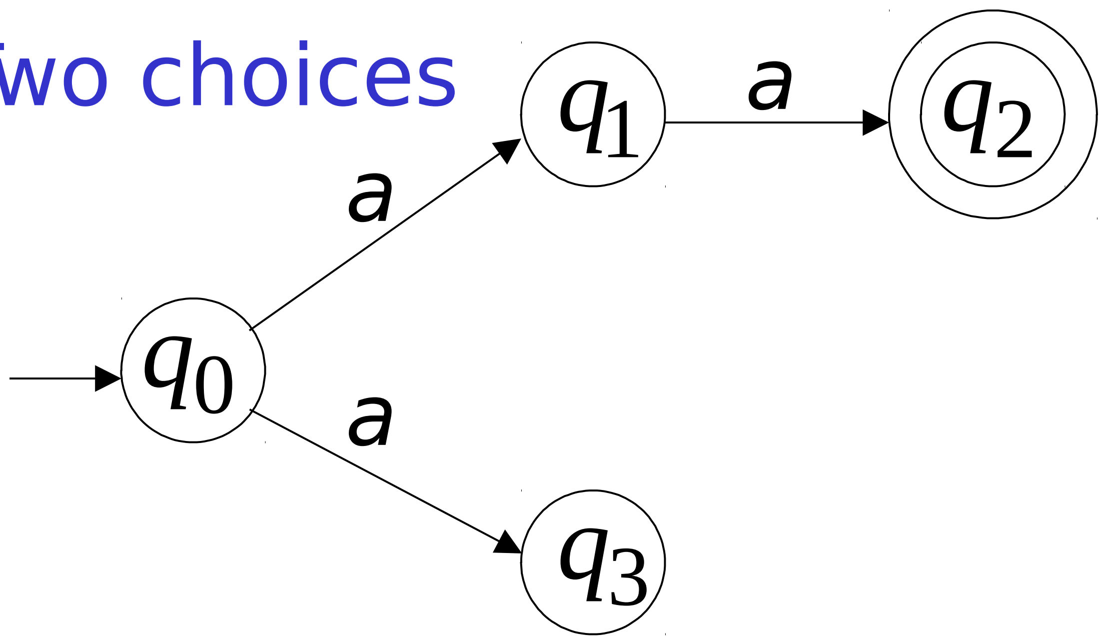
Alphabet $= \{a\}$



Nondeterministic Finite Accepter (NFA)

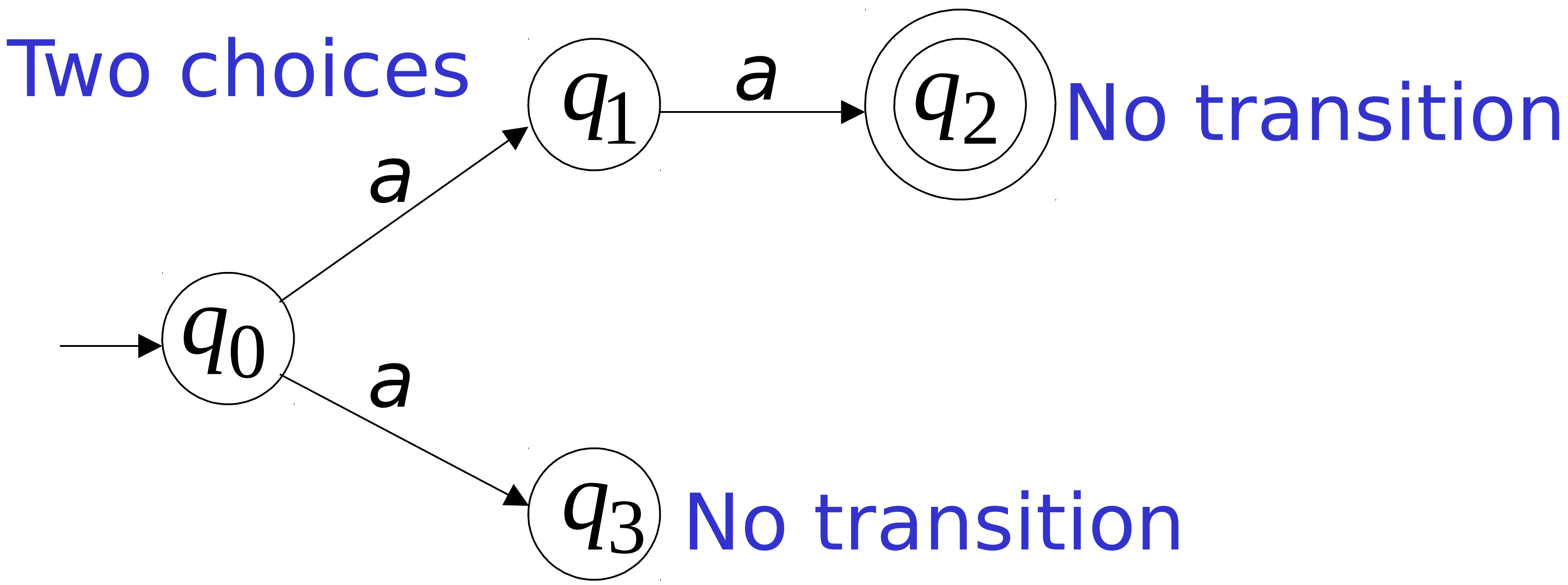
Alphabet = {a}

Two choices

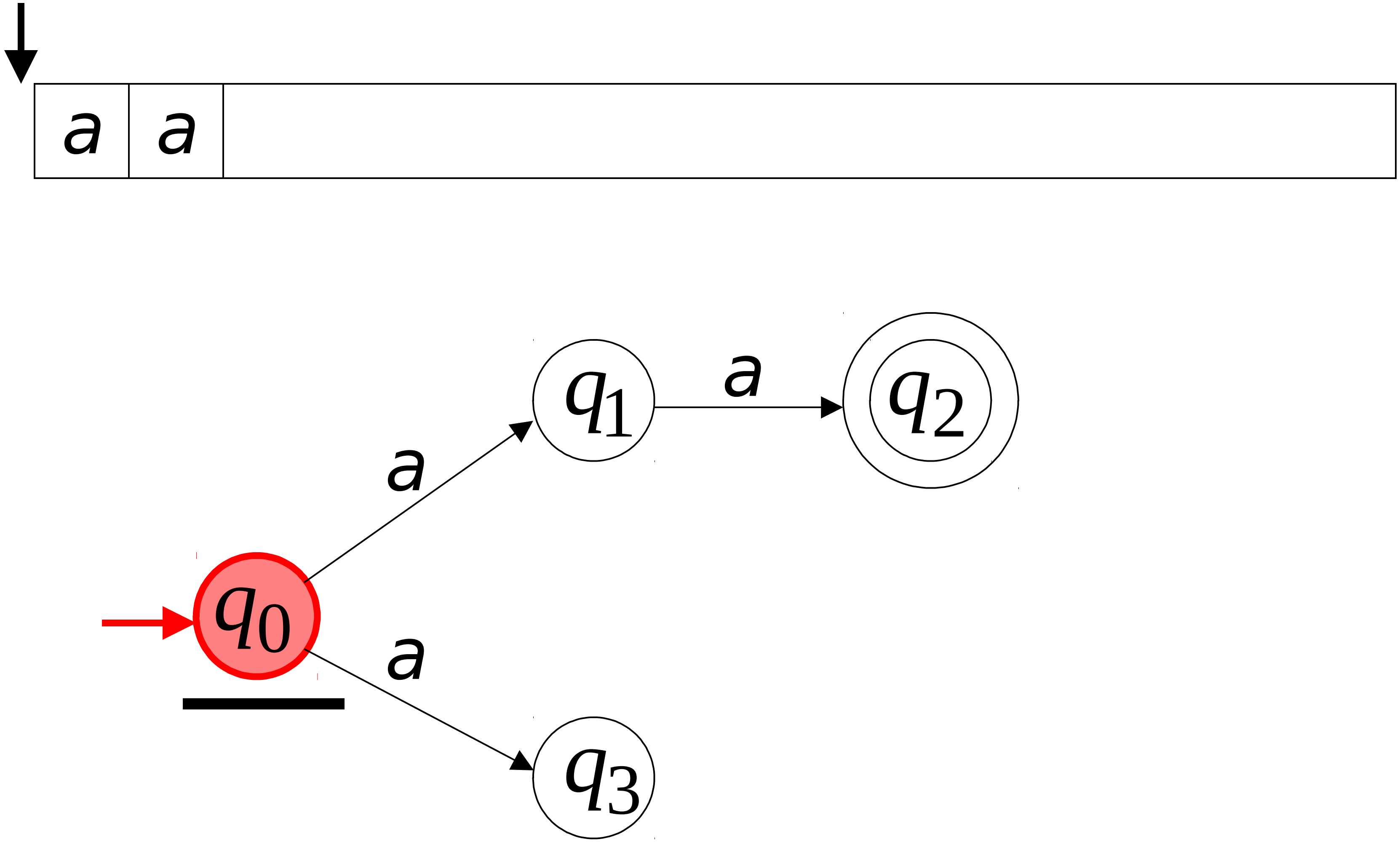


Nondeterministic Finite Accepter (NFA)

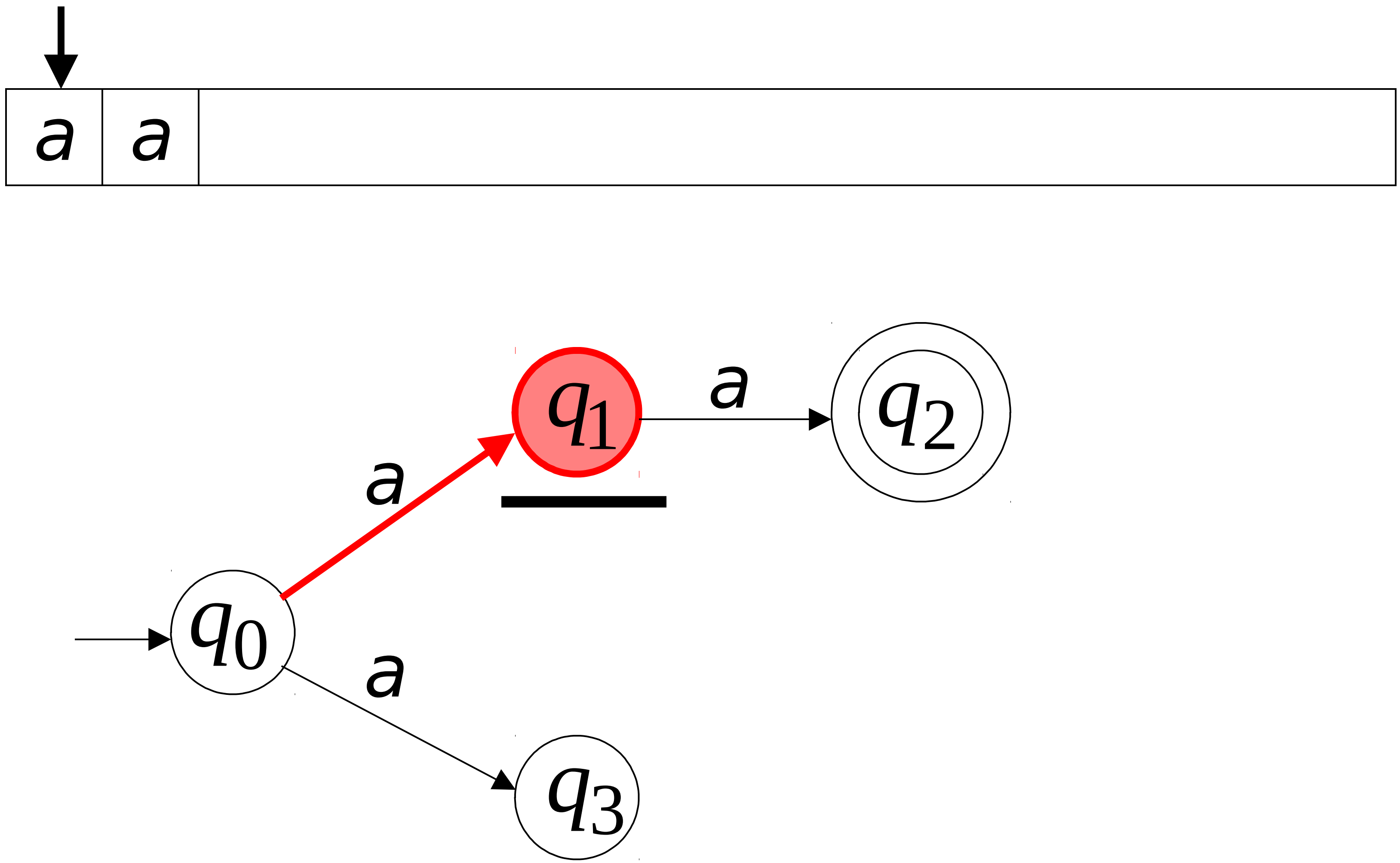
Alphabet $=\{a\}$



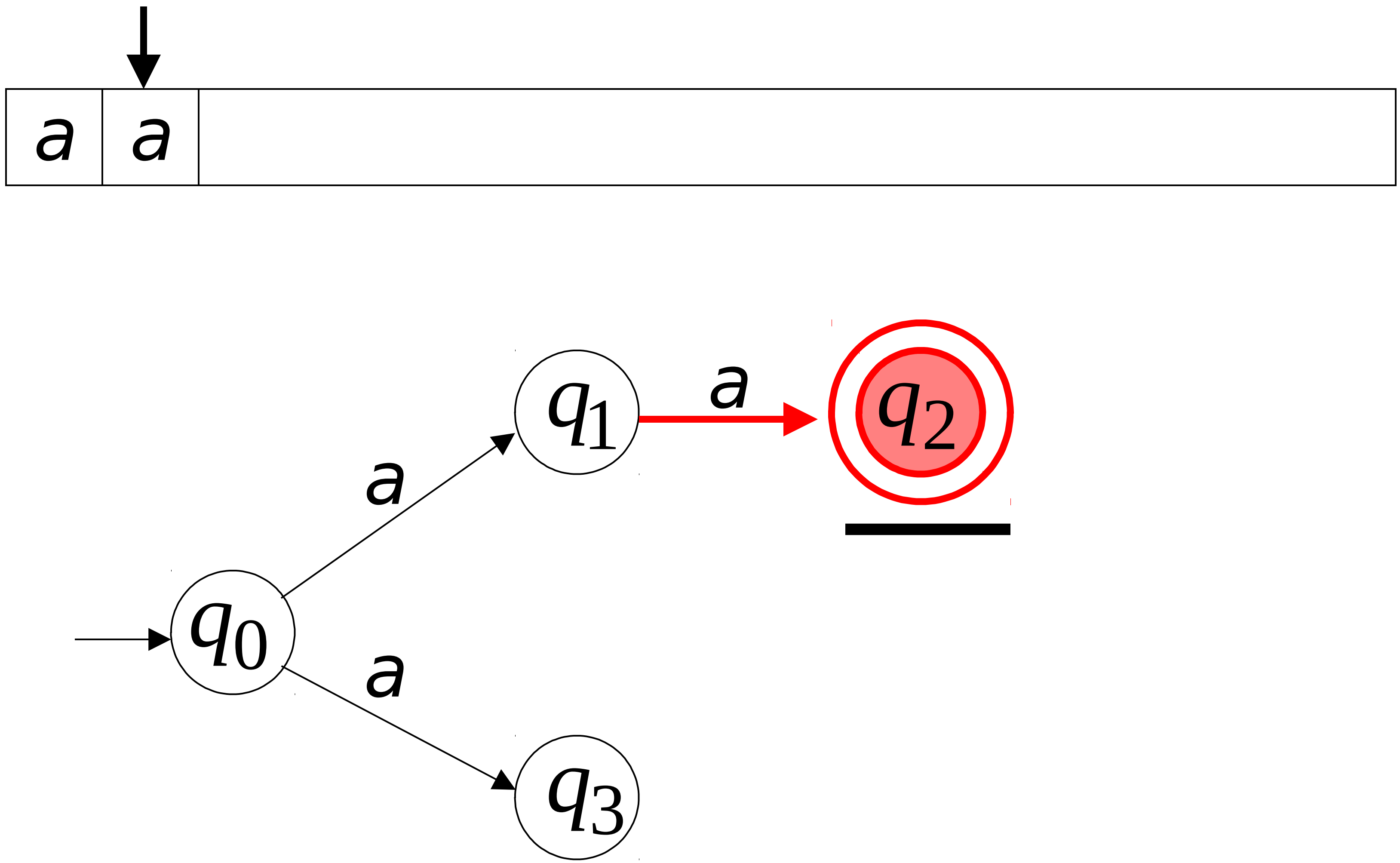
First Choice



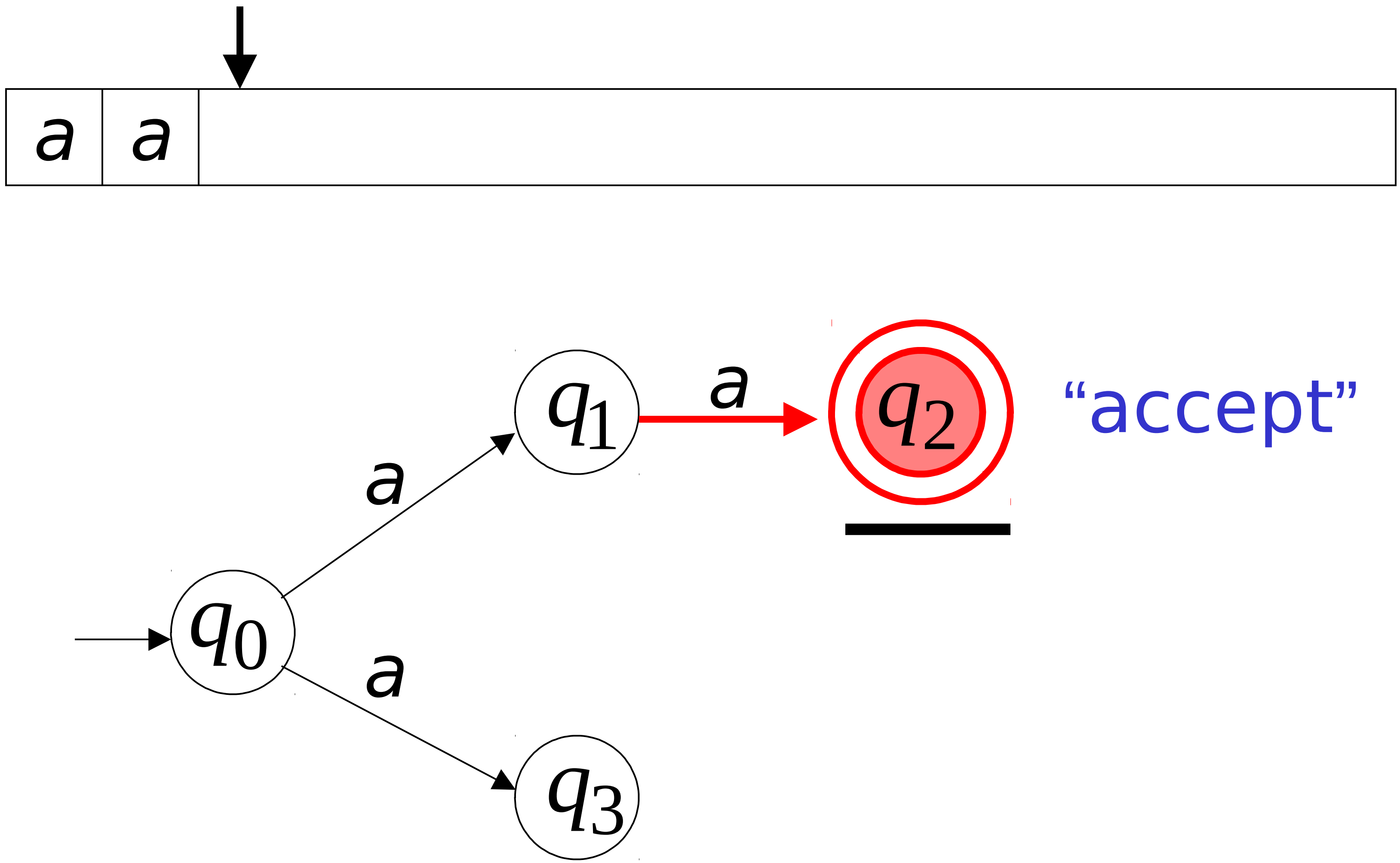
First Choice



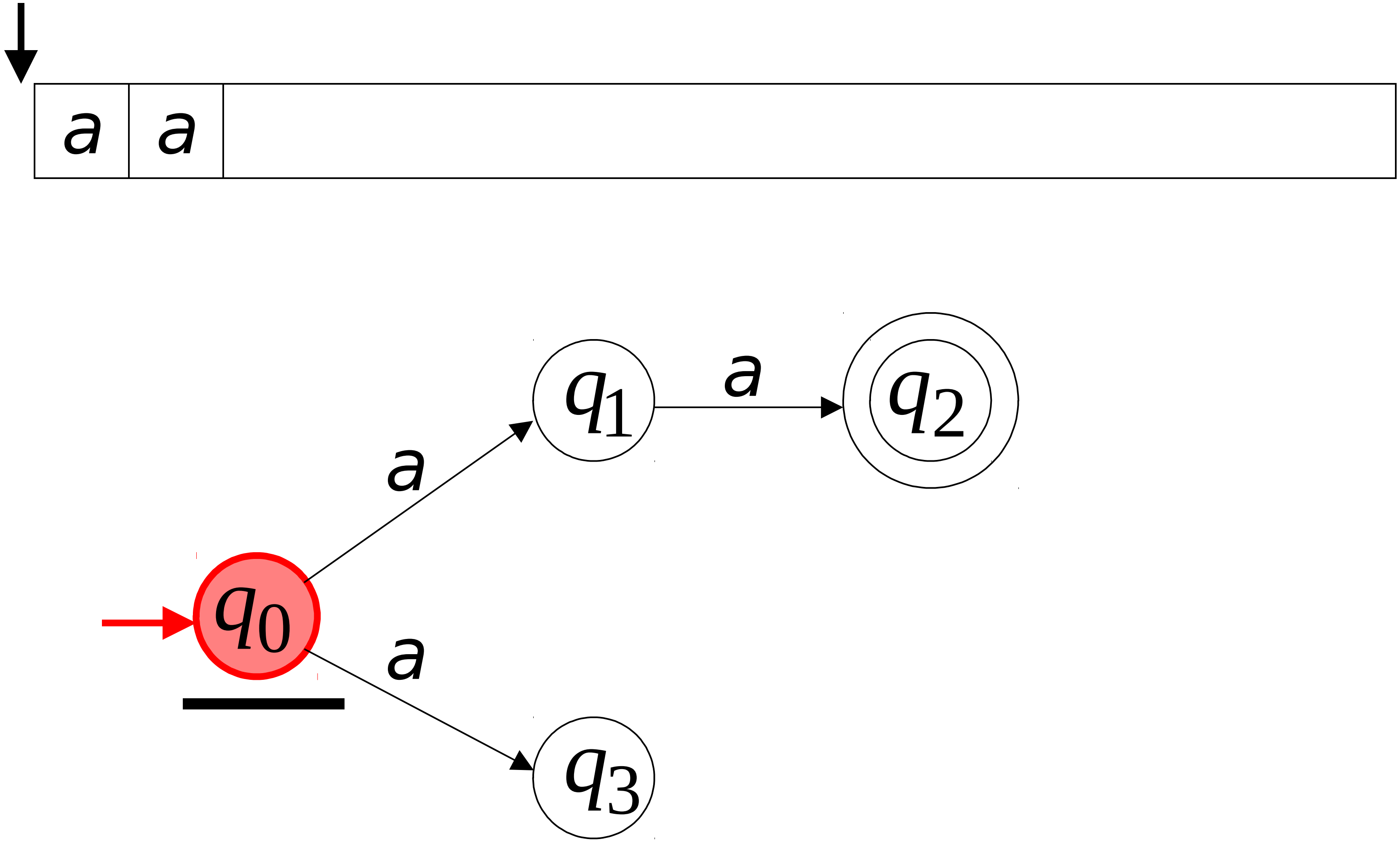
First Choice



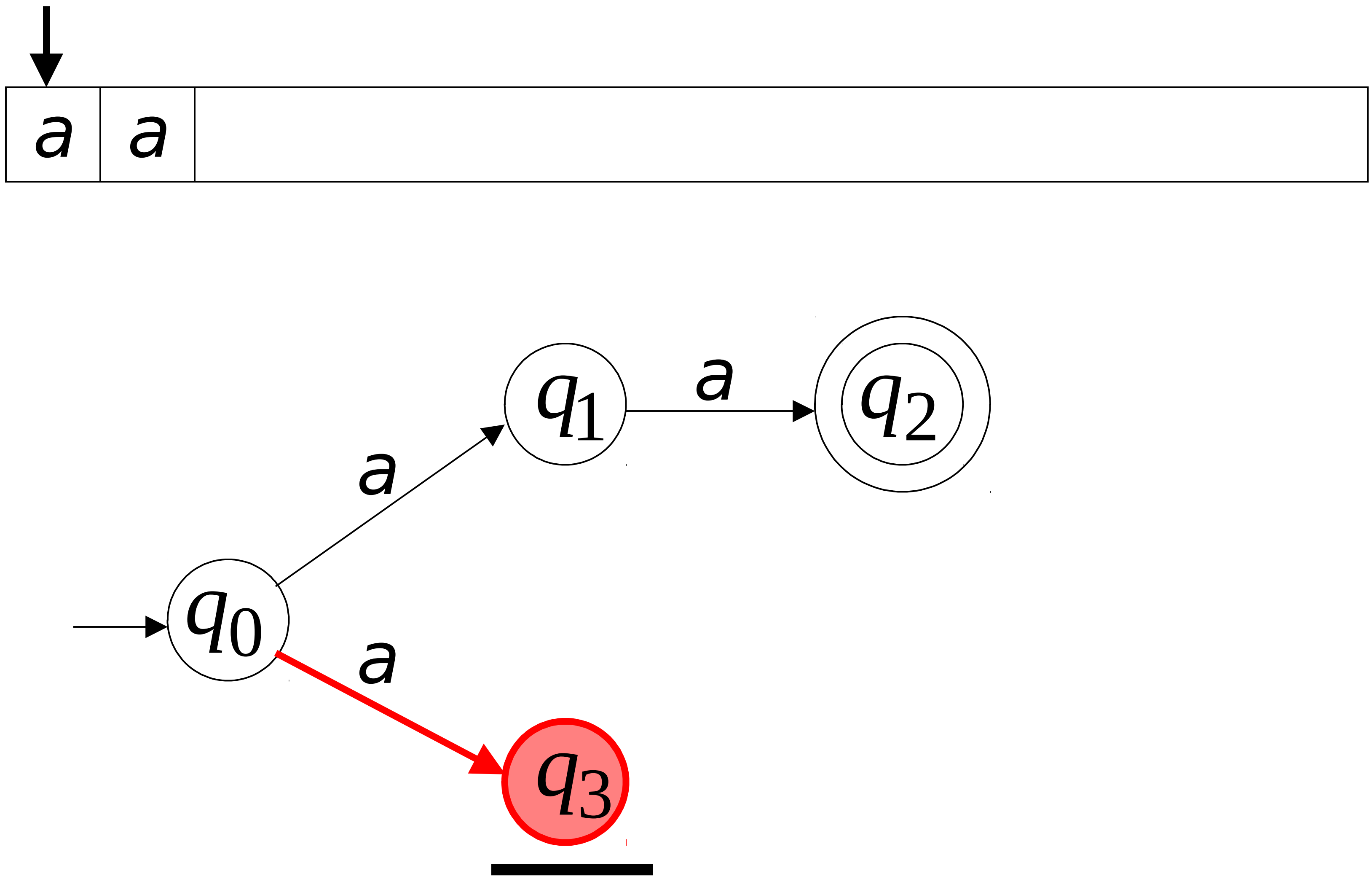
First Choice



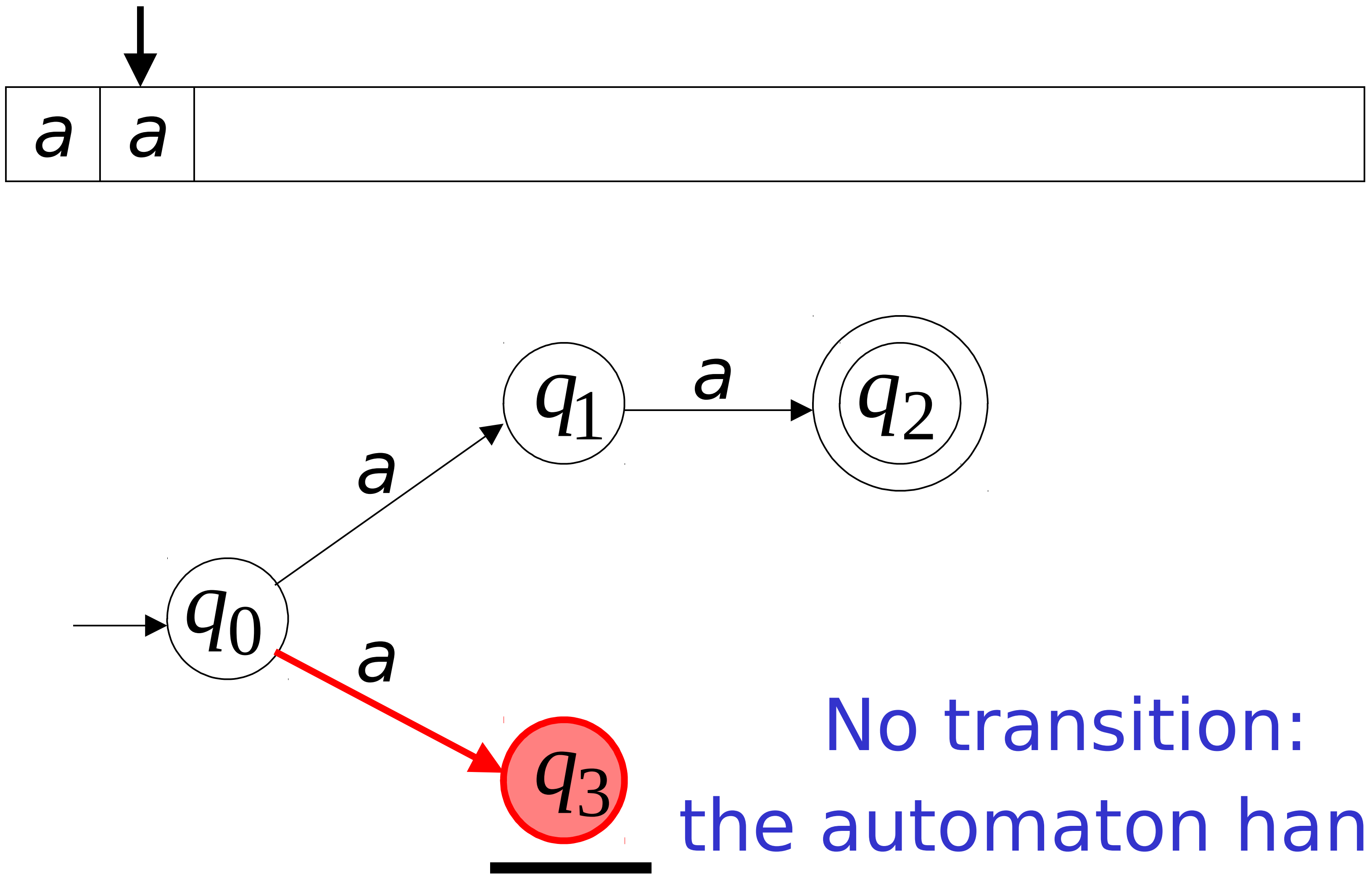
Second Choice



Second Choice

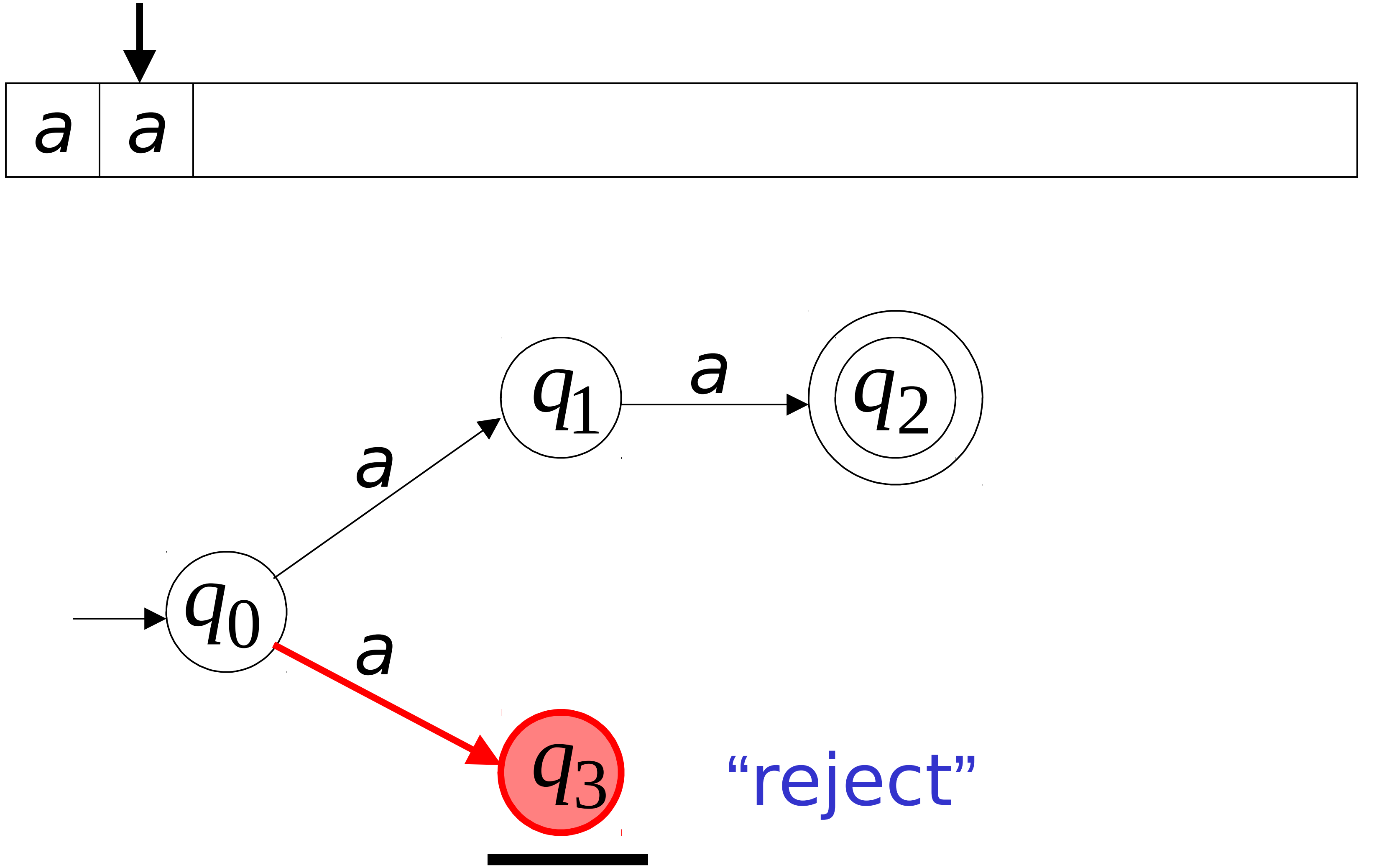


Second Choice



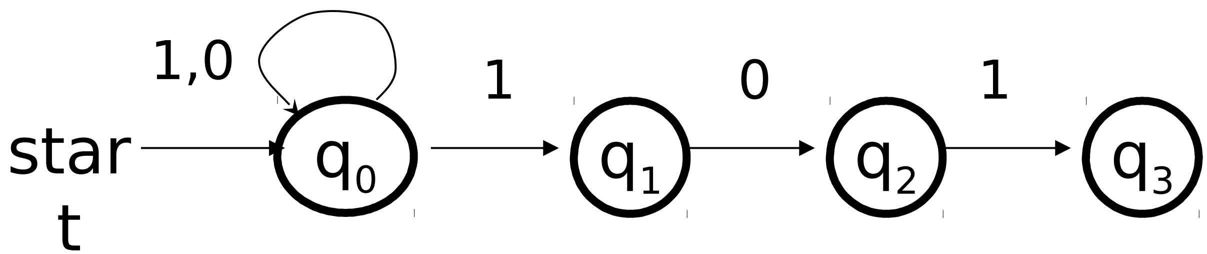
No transition:
the automaton hangs

Second Choice

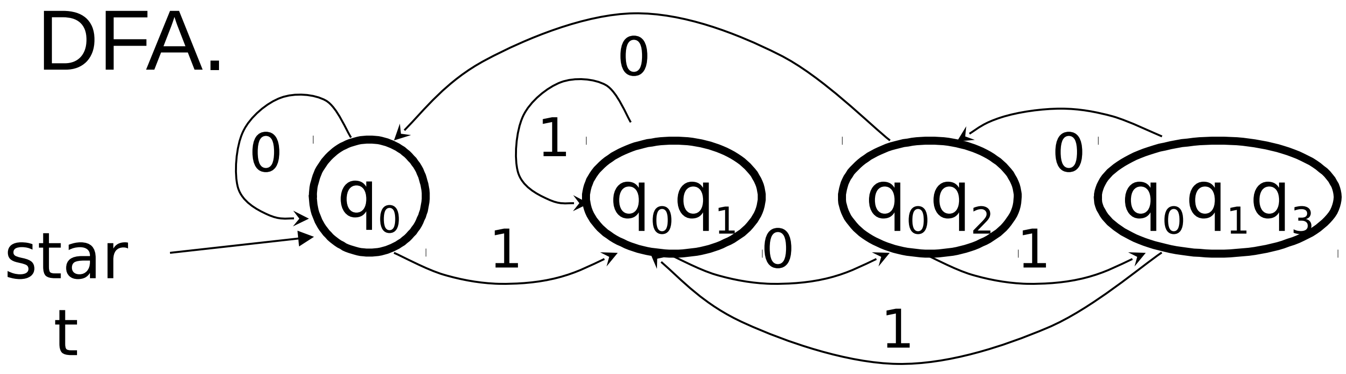


NFA for Common Suffix

- We can have a simpler representation for common suffix language using NFA:

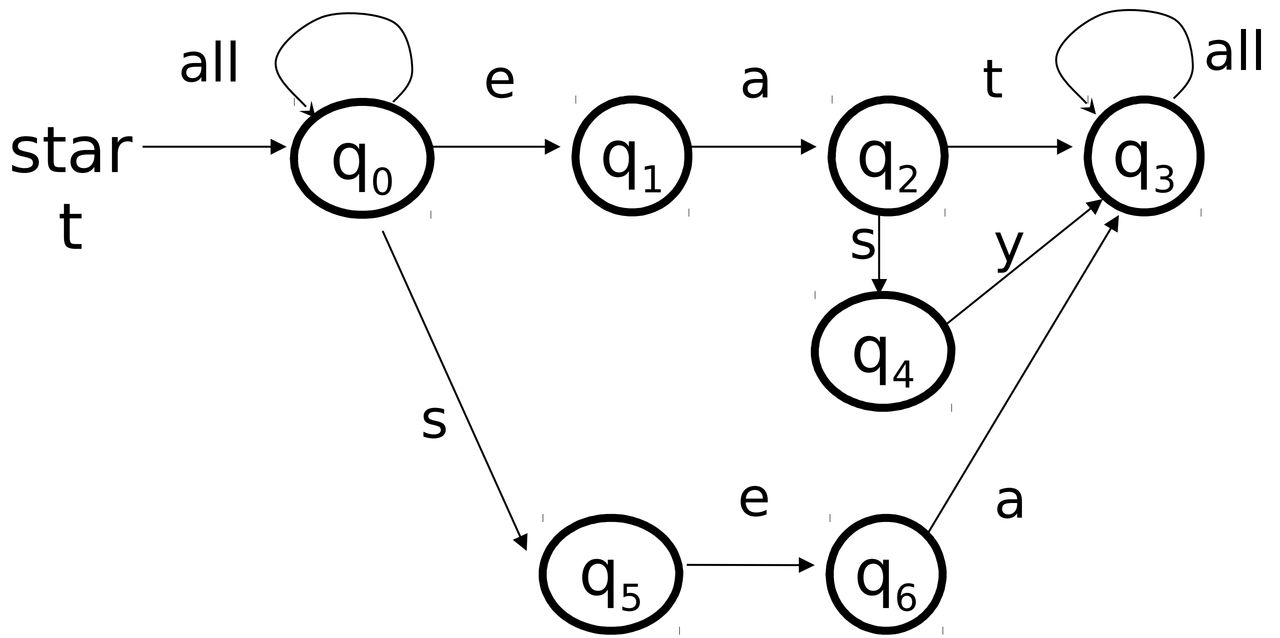


- Use subset construction to convert it to a DFA.



NFA Example

- Construct NFAs for the following languages over the alphabet {a, b, …, z}:
  - All strings that contain eat or sea or easy



Subset Construction Method

Using Subset construction method to convert NFA to DFA involves the following steps:

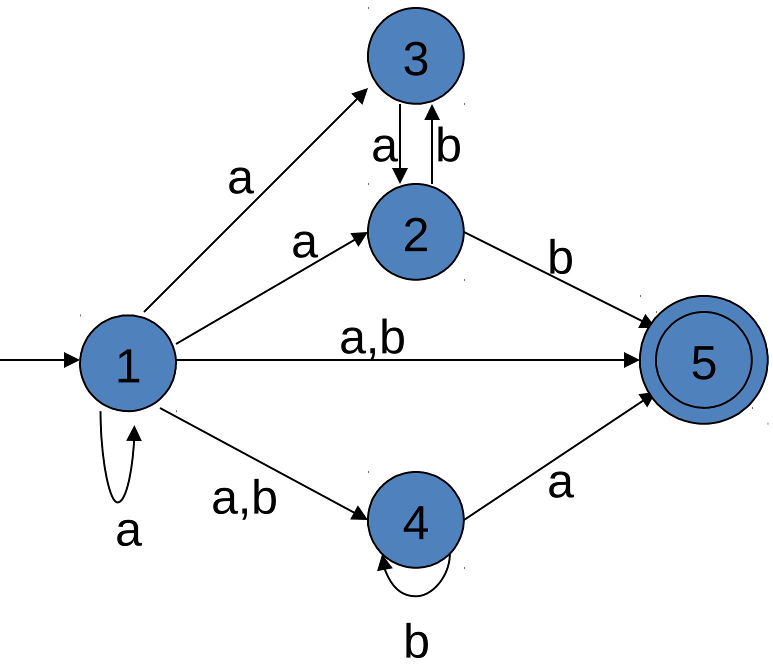
- For every state in the NFA, determine all *reachable states* for every input symbol.
- The set of reachable states constitute a *single state* in the converted DFA (Each state in the DFA corresponds to a subset of states in the NFA).
- Find *reachable states* for each new DFA state, until no more new states can be found.

Subset Construction Method

Fig1. NFA without λ-transitions

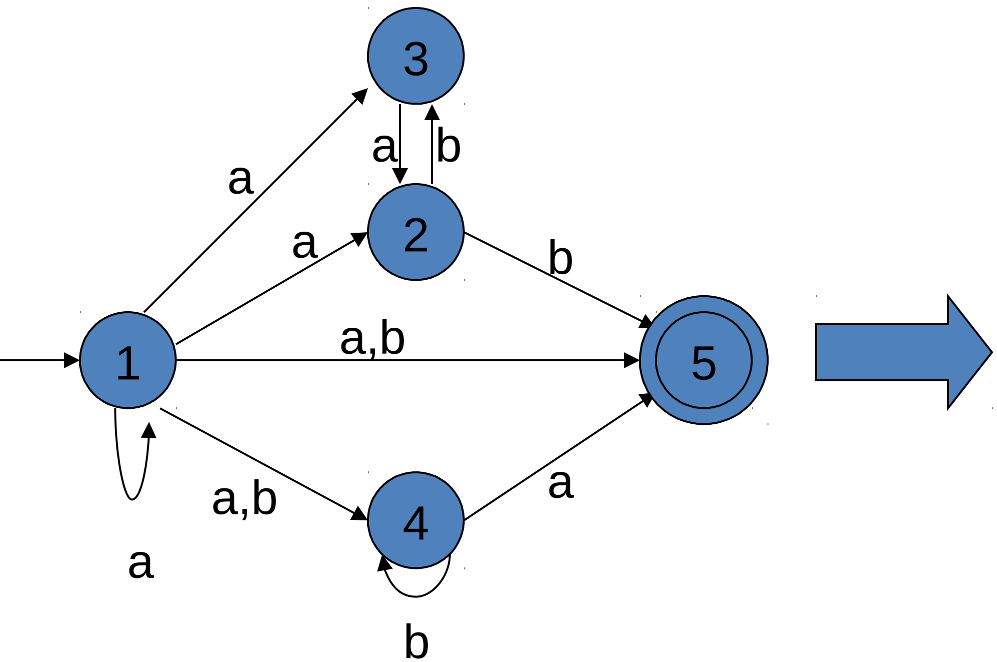
Subset Construction Method

Fig1. NFA without λ-transitions



Subset Construction Method

Fig1. NFA without λ-transitions



Step1

Construct a transition table showing all reachable states for every state for every input signal.

Subset Construction Method

Fig1. NFA without λ-transitions

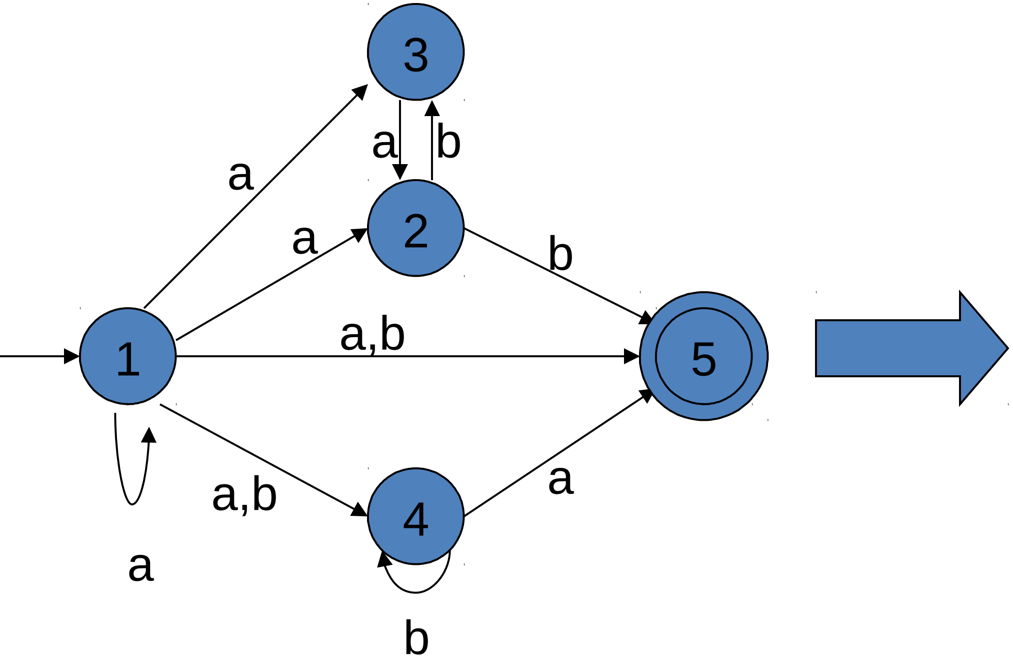


Fig2. Transition table

Subset Construction Method

Fig1. NFA without λ-transitions

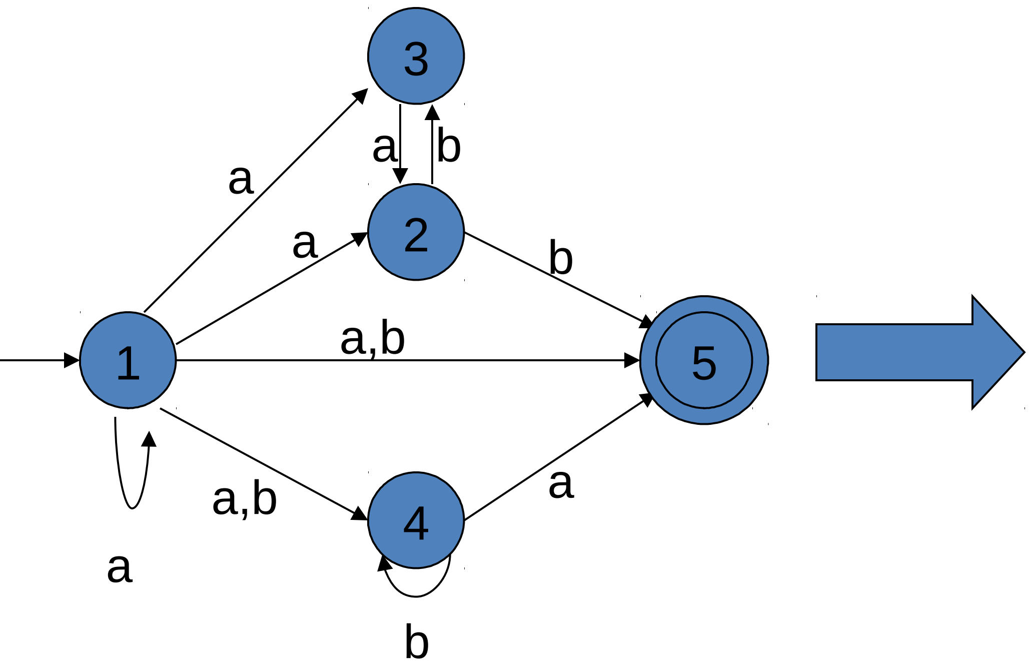
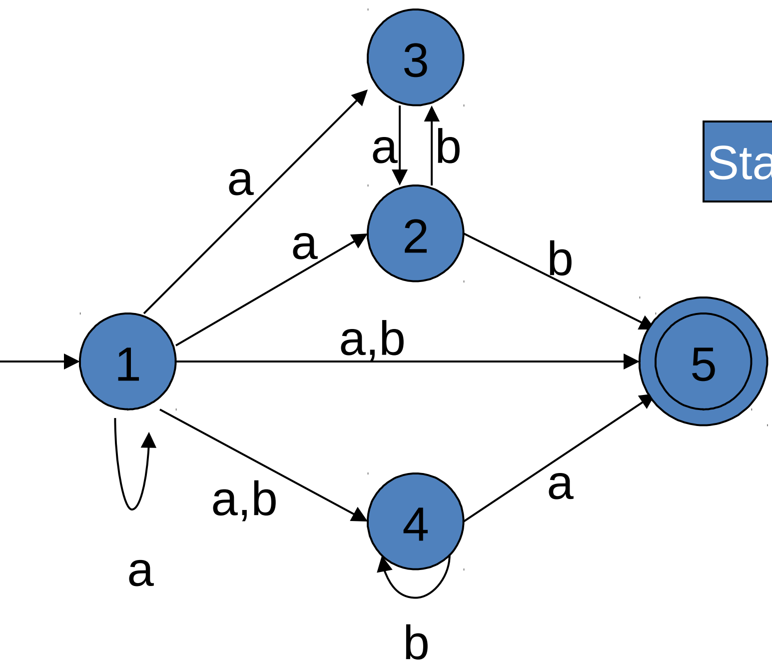


Fig2. Transition table

q	$\delta(q,a)$	$\delta(q,b)$
1	{1,2,3,4,5}	{4,5}
2	---	{3,5}
3	{2}	---
4	{5}	{4}
5	---	---

Subset Construction Method

Fig1. NFA without λ-transitions



Transition from state q with input a

Transition from state q with input b

Starts here

q	$\delta(q,a)$	$\delta(q,b)$
1	{1,2,3,4,5}	{4,5}
2	--	{3,5}
3	{2}	--
4	{5}	{4}
5	---	---

Subset Construction Method

Fig2. Transition table

q	δ(q,a)	δ(q,b)
1	{1,2,3,4,5}	{4,5}
2	–-	{3,5}
3	{2}	---
4	{5}	{4}
5	---	---

Step2

The set of states resulting from every transition function constitutes a new state. Calculate all reachable states for every such state for every input signal.

Fig2. Transition table

q	δ(q,a)	δ(q,b)
1	{1,2,3,4,5}	{4,5}
2	---	{3,5}
3	{2}	---
4	{5}	{4}
5	---	---

Starts with
Initial state

Fig3. Subset Construction table

q	δ(q,a)	δ(q,b)
{1 }	{1,2,3,4,5}	{4,5}

Fig2. Transition table

q	δ(q,a)	δ(q,b)
1	{1,2,3,4,5}	{4,5}
2	---	{3,5}
3	{2}	---
4	{5}	{4}
5	---	---

Starts with
Initial state

Fig3. Subset Construction table

q	δ(q,a)	δ(q,b)
{ 1}	{1,2,3,4,5}	{4,5}
{1,2,3,4,5}		
{4,5}		

Fig2. Transition table

q	δ(q,a)	δ(q,b)
1	{1,2,3,4,5}	{4,5}
2	---	{3,5}
3	{2}	---
4	{5}	{4}
5	---	---

Starts with Initial state

Fig3. Subset Construction table

q	δ(q,a)	δ(q,b)
{ 1}	{1,2,3,4,5}	{4,5}
{1,2,3,4,5}		
{4,5}		

Step3

Repeat this process(step2) until no more new states are reachable.

Fig2. Transition table

q	δ(q,a)	δ(q,b)
1	{1,2,3,4,5}	{4,5}
2	–-	{3,5}
3	{2}	---
4	{5}	{4}
5	---	---

Fig3. Subset Construction table

q	δ(q,a)	δ(q,b)
{ 1}	{1,2,3,4,5}	{4,5}
{1,2,3,4,5}	{1,2,3,4,5}	{3,4,5}
{4,5}		
{3,4,5}		

Fig2. Transition table

q	δ(q,a)	δ(q,b)
1	{1,2,3,4,5}	{4,5}
2	---	{3,5}
3	{2}	---
4	{5}	{4}
5	---	---

Fig3. Subset Construction table

q	δ(q,a)	δ(q,b)
{ 1}	{1,2,3,4,5}	{4,5}
{1,2,3,4,5}	{1,2,3,4,5}	{3,4,5}
{4,5}	{5}	{4}
{3,4,5}		
{5}		
{4}		

Fig2. Transition table

q	δ(q,a)	δ(q,b)
1	{1,2,3,4,5}	{4,5}
2	---	{3,5}
3	{2}	---
4	{5}	{4}
5	---	---

Fig3. Subset Construction table

q	δ(q,a)	δ(q,b)
{ 1}	{1,2,3,4,5}	{4,5}
{1,2,3,4,5}	{1,2,3,4,5}	{3,4,5}
{4,5}	{5}	{4}
{3,4,5}	{2,5}	{4}
{5}		
{4}		
{2,5}		

Fig2. Transition table

q	δ(q,a)	δ(q,b)
1	{1,2,3,4,5}	{4,5}
2	---	{3,5}
3	{2}	---
4	{5}	{4}
5	---	---

Fig3. Subset Construction table

q	δ(q,a)	δ(q,b)
{ 1}	{1,2,3,4,5}	{4,5}
{1,2,3,4,5}	{1,2,3,4,5}	{3,4,5}
{4,5}	{5}	{4 }
{3,4,5}	{2,5}	{4 }
{5}	$\emptyset$	$\emptyset$
{4 }		
{3,5}		
$\emptyset$		

Fig2. Transition table

q	δ(q,a)	δ(q,b)
1	{1,2,3,4,5}	{4,5}
2	---	{3,5}
3	{2}	---
4	{5}	{4}
5	---	---

Fig3. Subset Construction table

q	δ(q,a)	δ(q,b)
{ 1}	{1,2,3,4,5}	{4,5}
{1,2,3,4,5}	{1,2,3,4,5}	{3,4,5}
{4,5}	{5}	{4 }
{3,4,5}	{2,5}	{4 }
{5}	∅	∅
{4 }	{5}	{4 }
{2,5}		
∅		

We already got 4 and 5.
So we don't add them again.

Fig2. Transition table

q	δ(q,a)	δ(q,b)
1	{1,2,3,4,5}	{4,5}
2	---	{3,5}
3	{2}	---
4	{5}	{4}
5	---	---

Fig3. Subset Construction table

q	δ(q,a)	δ(q,b)
{ 1}	{1,2,3,4,5}	{4,5}
{1,2,3,4,5}	{1,2,3,4,5}	{3,4,5}
{4,5}	{5}	{4 }
{3,4,5}	{2,5}	{4 }
{5}	∅	∅
{4 }	{5}	{4 }
{2,5}	---	{3,5}
∅		
{3,5}		

Fig2. Transition table

q	δ(q,a)	δ(q,b)
1	{1,2,3,4,5}	{4,5}
2	---	{3,5}
3	{2}	---
4	{5}	{4}
5	---	---

Fig3. Subset Construction table

q	δ(q,a)	δ(q,b)
{ 1}	{1,2,3,4,5}	{4,5}
{1,2,3,4,5}	{1,2,3,4,5}	{3,4,5}
{4,5}	{5}	{4 }
{3,4,5}	{2,5}	{4 }
{5}	∅	∅
{4 }	{5}	{4 }
{2,5}	---	{3,5}
∅	∅	∅
{3,5}	{2}	---

Fig2. Transition table

q	δ(q,a)	δ(q,b)
1	{1,2,3,4,5}	{4,5}
2	---	{3,5}
3	{2}	---
4	{5}	{4}
5	---	---

Fig3. Subset Construction table

q	δ(q,a)	δ(q,b)
{ 1}	{1,2,3,4,5}	{4,5}
{1,2,3,4,5}	{1,2,3,4,5}	{3,4,5}
{4,5}	{5}	{4 }
{3,4,5}	{2,5}	{4 }
{5}	$\varnothing$	$\varnothing$
{4 }	{5}	{4 }
{2,5}	---	{3,5}
$\varnothing$	$\varnothing$	$\varnothing$
{3,5}	{2}	---
{2}	---	{3,5}

Fig2. Transition table

q	δ(q,a)	δ(q,b)
1	{1,2,3,4,5}	{4,5}
2	---	{3,5}
3	{2}	---
4	{5}	{4}
5	---	---

Fig3. Subset Construction table

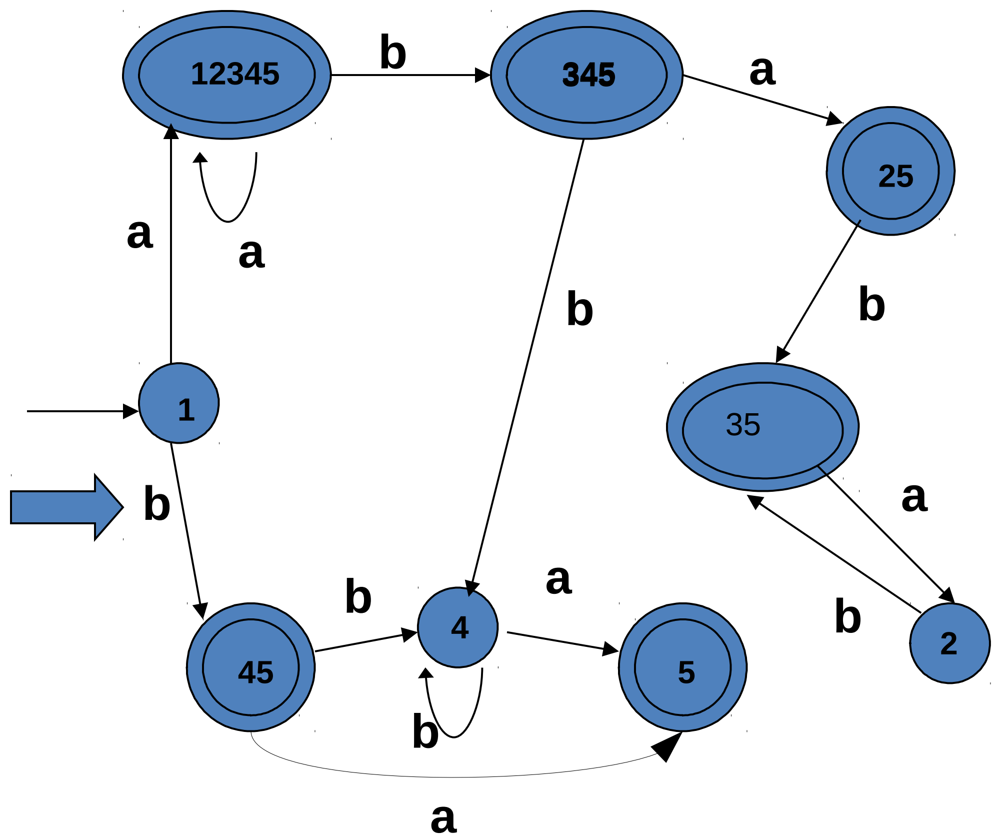
q	δ(q,a)	δ(q,b)
{ 1}	{1,2,3,4,5}	{4,5}
{1,2,3,4,5}	{1,2,3,4,5}	{3,4,5}
{4,5}	{5}	{4 }
{3,4,5}	{2,5}	{4 }
{5}	∅	∅
{4 }	{5}	{4 }
{2,5}	---	{3,5}
∅	∅	∅
{3,5}	{2}	---
{2}	---	{3,5}

Stops here as there are no more reachable states

Fig3. Subset Construction table

q	δ(q,a)	δ(q,b)
{ 1}	{1,2,3,4,5}	{4,5}
{1,2,3,4,5}	{1,2,3,4,5}	{3,4,5}
{4,5}	{5}	{4 }
{3,4,5}	{2,5}	{4 }
{5}	∅	∅
{4 }	{5}	{4 }
{2,5}	---	{3,5}
∅	∅	∅
{3,5}	{2}	---
{2}	---	{3,5}

Fig4. Resulting FA after applying Subset Construction to fig1



Extended Transition Function

Extended transition function $\hat{\delta}$ takes a state ***q*** and a string of input symbols ***w***, and returns the set of states that the NFA is in if it starts in state ***q*** and processes the string ***w***.

$\hat{\delta}(q,w) = \{ q_k \}$

$\hat{\delta}(q,\varepsilon) = q$, $\hat{\delta}(q,a) = \delta(\hat{\delta}(q,\varepsilon),a) = \delta(q,a)$

Let w=xa and If $\hat{\delta}(qi,x) = q_k$ and $\delta(q_k,a) = q_j$

Then $\hat{\delta}(qi,xa) = q_j$

Ie $\hat{\delta}(qi,w) = q_j$

Language of an NFA

If $M = (Q, \Sigma, \delta, q0, F)$ is an NFA, then language accepted by NFA $L(M)$ can be defined as

$$L(M) = \{ w \mid \hat{\delta}(q0,w) \cap F \neq \emptyset \}$$

Theorem :

If $D = (Q_D, \Sigma, \delta_D, \{q0\}, F_D)$ is the DFA constructed from NFA $N = (Q_N, \Sigma, \delta_N, q0, F_N)$ by the subset construction, then $L(D) = L(N)$.

OR

Let the language L be a set accepted by an NFA, then there exists an equivalent DFA that accepts L.

Proof

δ_D, the transition function for D is defined as $\delta_D(q,a) = \cup_{p \in q}\, \delta_N(p,a)$ for $q \in Q_D$ and $a \in \Sigma$.

F_D is the set of states in Q_D that contain any element of F_N, $F_D = \{q \in Q_D | q \cap F_N \neq \emptyset\}$

<u>Basis step</u> : Let x be the empty string ε.

$\hat{\delta}_N(q_0,\varepsilon) = \{q_0\} = \hat{\delta}_D(\{q_0\},\varepsilon)$; by the definition of $\hat{\delta}$

Induction : Let us assume that the theory is true for string x of length n or less.

ie. $\hat{\delta_N}(q_0,x) = \hat{\delta_D}(\{q_0\},x) = \{p1,p2,....,pk\}$; by the inductive hypothesis.

Let w = xa be a string of length n+1 with 'a' in Σ.

$\hat{\delta_N}(q_0,w) = U_{i=1}^{k} \delta_N(pi,a)$ ie $\delta_N(\{p1,p2,...,pk\},a)$

By subset construction

$\hat{\delta_D}(\{q_0\},w) = \hat{\delta_D}(\{q_0\},xa)$

$= \delta_D(\hat{\delta_D}(\{q_0\},x),a)$

$= \delta_D(\{p1,p2,....,pk\},a)$

$= U_{i=1}^{k} \delta_N(pi,a)$

$= \hat{\delta_N}(q_0,w)$

w is accepted by D only if $\hat{\delta_D}(\{q_0\},w) \in F_D$. Which contains a state in F_N. Therefor w is also accepted by N.

ie L(N) = L(D)

Regular Expressions

Regular expressions

- A regular expression over Σ is an expression formed using the following rules:
  - The symbol $\emptyset$ is a regular expression
  - The symbol ε is a regular expression
  - For every $a \in \Sigma$, the symbol a is a regular expression
  - If R and S are regular expressions, so are $R+S$, RS and R^*.

A language is regular if it is represented by a regular expression

Regular Expressions

Regular expressions describe regular languages

Example:

$$(a + b \cdot c)^*$$

describes the language

$$\{a, bc\}^* = \{\lambda, a, bc, aa, abc, bca, \ldots\}$$

Recursive Definition

Primitive regular expressions: $\emptyset, \quad \lambda, \quad a$

Given regular expressions r_1 and r_2

$$\left.\begin{array}{l} r_1 + r_2 \\ r_1 \cdot r_2 \\ r_1 * \\ (r_1) \end{array}\right\} \text{Are regular expressions}$$

Examples

A regular expression: $(a+b\cdot c)*\cdot(c+\emptyset)$

Not a regular expression: $(a+b+)$

Regular Expressions

- Alphabet $X = \{a, b, c, d\}$

1) $(a + b)^*$

all strings containing only a and b

2) $c(a + b + c)^*c^2$

all strings containing only a, b, and c that begin with c and end with cc

3) all strings containing only one b

$(a+c+d)^*b(a+c+d)^*$

Regular Expressions

- Alphabet X = {0, 1}

    1) $(0 + 1)^*$

    the set of all binary strings

    2) $0^3 1^* 0^4$

    all strings consisting of three 0’s, followed by any number of 1’s, followed by four 0’s

    3) $0^* 1001^*$

    all strings starting with any number of 0’s, followed by 100, followed by and number of 1’s

Regular Languages

- A *regular language* is one that has a DFA that accepts it
- We proved that the *complement* of a regular language is also regular!
- To show that a language is regular, we find a DFA for it
- If it isn't regular, well, that's another story

Languages of Regular Expressions

$L(r)$: language of regular expression r

Example

$$L((a+b\cdot c)^*) = \{\lambda, a, bc, aa, abc, bca, \ldots\}$$

Definition

For primitive regular expressions:

$$L(\varnothing) = \varnothing$$

$$L(\lambda) = \{\lambda\}$$

$$L(a) = \{a\}$$

Definition (continued)

For regular expressions r_1 and r_2

$$L(r_1 + r_2) = L(r_1) \cup L(r_2)$$

$$L(r_1 \cdot r_2) = L(r_1)\, L(r_2)$$

$$L(r_1 *) = (L(r_1)) *$$

$$L((r_1)) = L(r_1)$$

Example

Regular expression: $(a+b)\cdot a*$

$$
\begin{aligned}
L((a+b)\cdot a*) &= L((a+b))\,L(a*)\\
&= L(a+b)\,L(a*)\\
&= (L(a)\cup L(b))\,(L(a))*\\
&= (\{a\}\cup\{b\})\,(\{a\})*\\
&= \{a,b\}\,\{\lambda,a,aa,aaa,\ldots\}\\
&= \{a,aa,aaa,\ldots,b,ba,baa,\ldots\}
\end{aligned}
$$

Some Properties of Regular Languages

We Say:

Regular languages are **closed under**

Union: $L_1 \cup L_2$

Concatenation: $L_1 L_2$

Star: $L_1 *$

Properties

For regular languages L_1 and L_2 we will prove that:

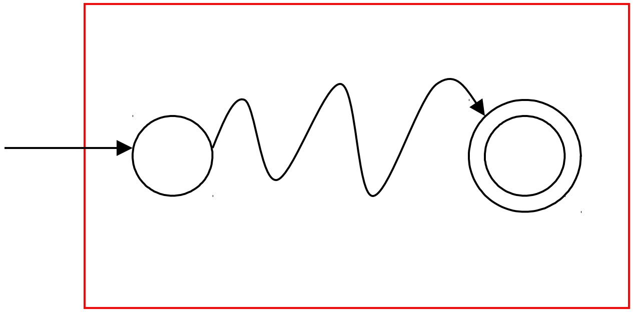
$$\left.\begin{array}{rl}\text{Union:} & L_1 \cup L_2 \\[2em] \text{Concatenation:} & L_1 L_2 \\[2em] \text{Star:} & L_1 *\end{array}\right\}\ \begin{array}{l}\text{Are regular}\\ \text{Languages}\end{array}$$

Regular language L_1 Regular language L_2

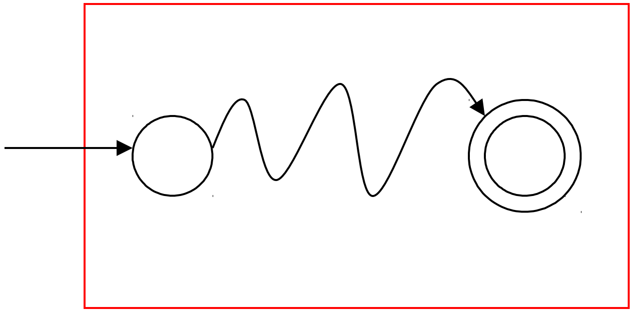
$$L(M_1) = L_1$$

$$L(M_2) = L_2$$

NFA M_1



NFA M_2

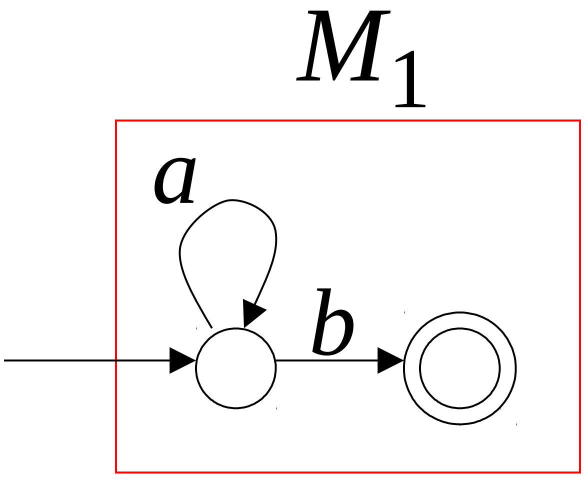


Single final state

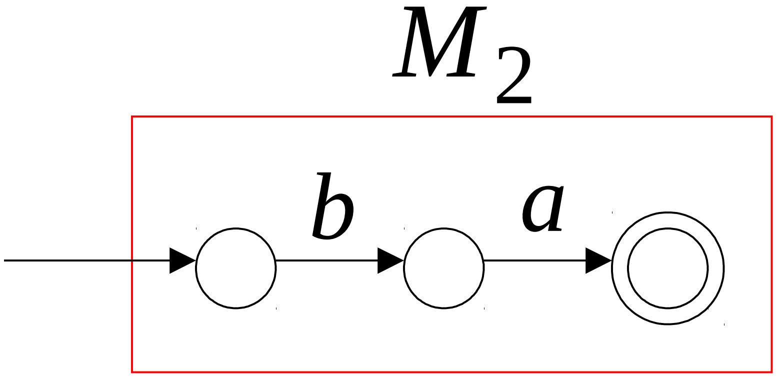
Single final state

Example

$$L_1 = \{a^n b\}$$

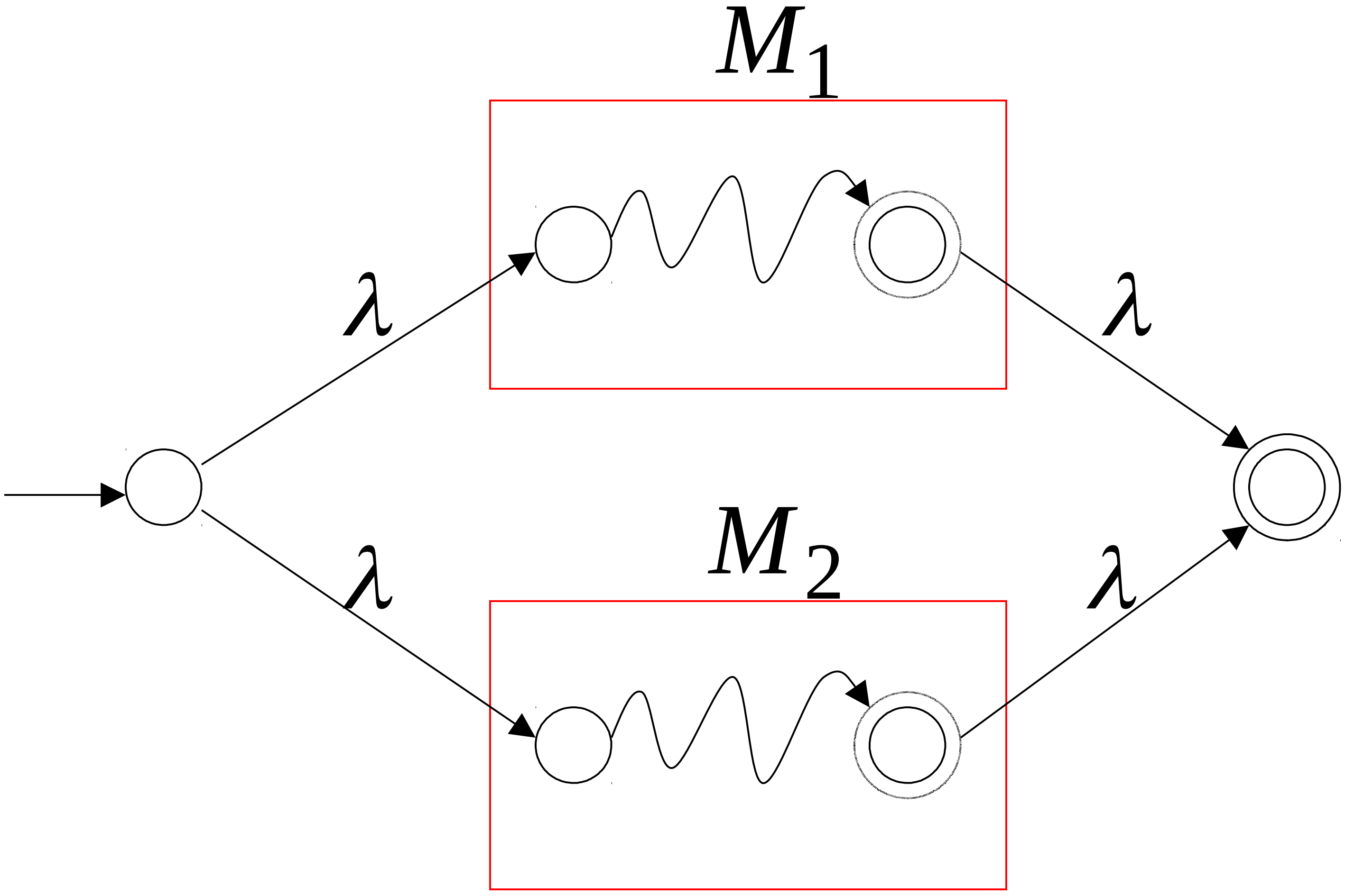


$$L_2 = \{ba\}$$



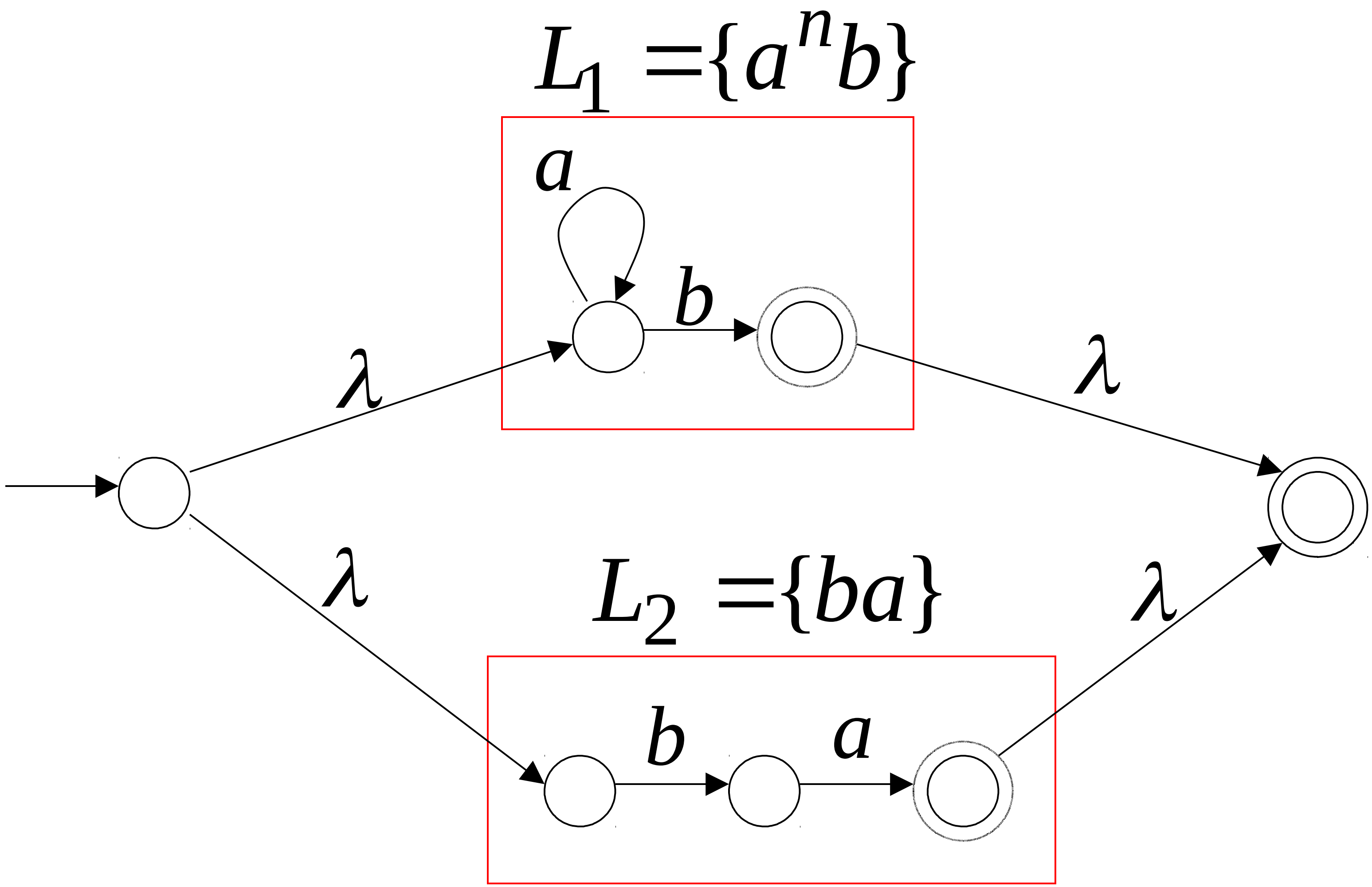
Union

NFA for $L_1 \cup L_2$



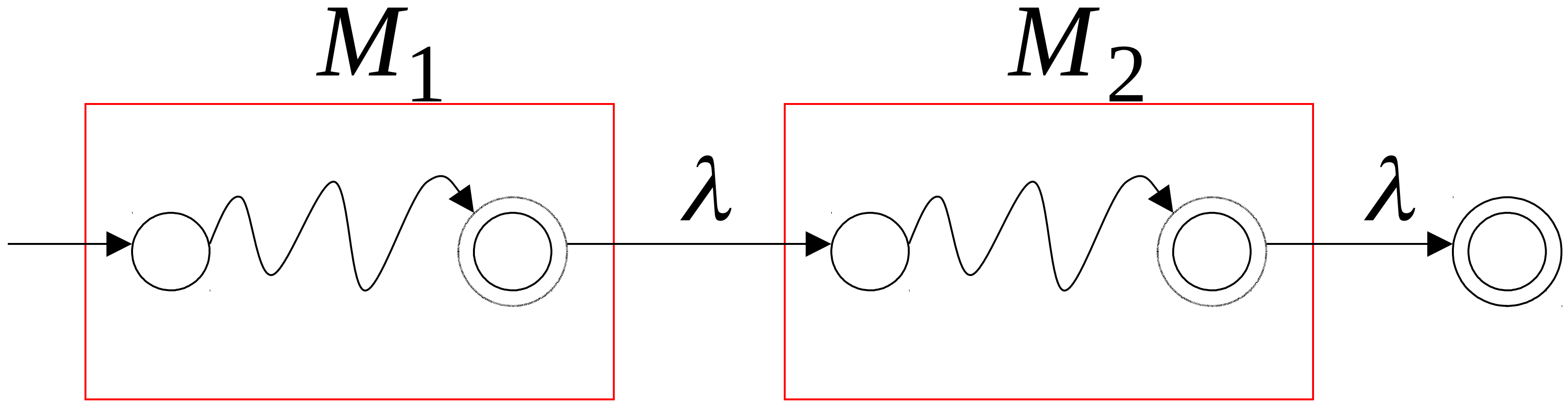
Example

NFA for $L_1 \cup L_2 = \{a^n b\} \cup \{ba\}$



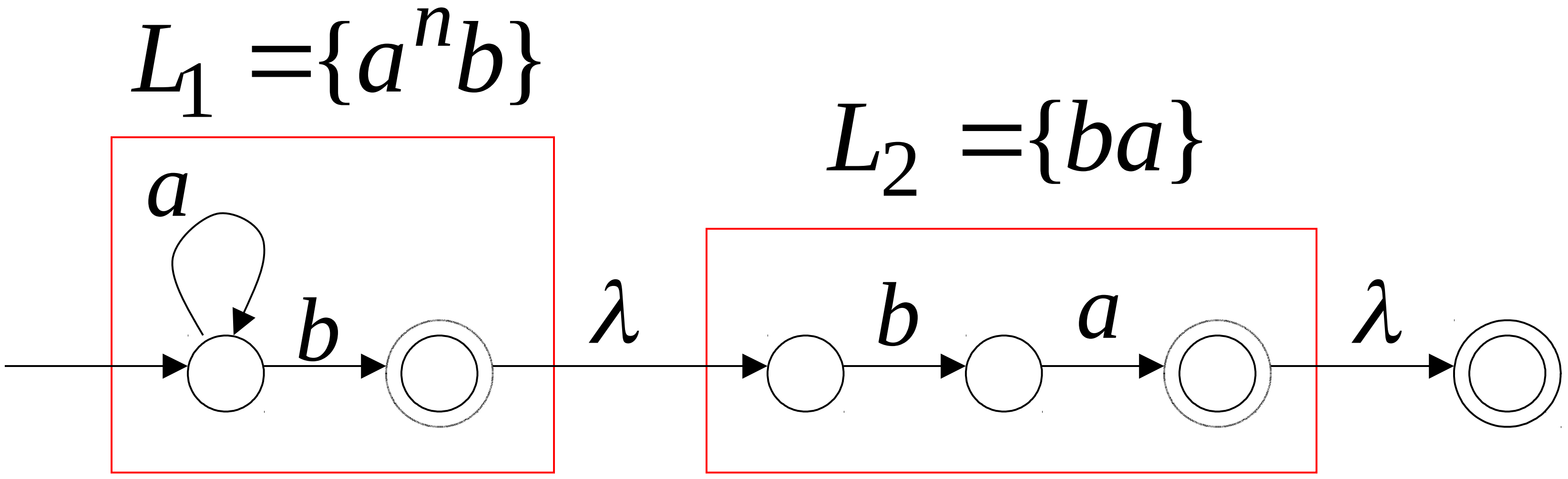
Concatenation

NFA for L_1L_2



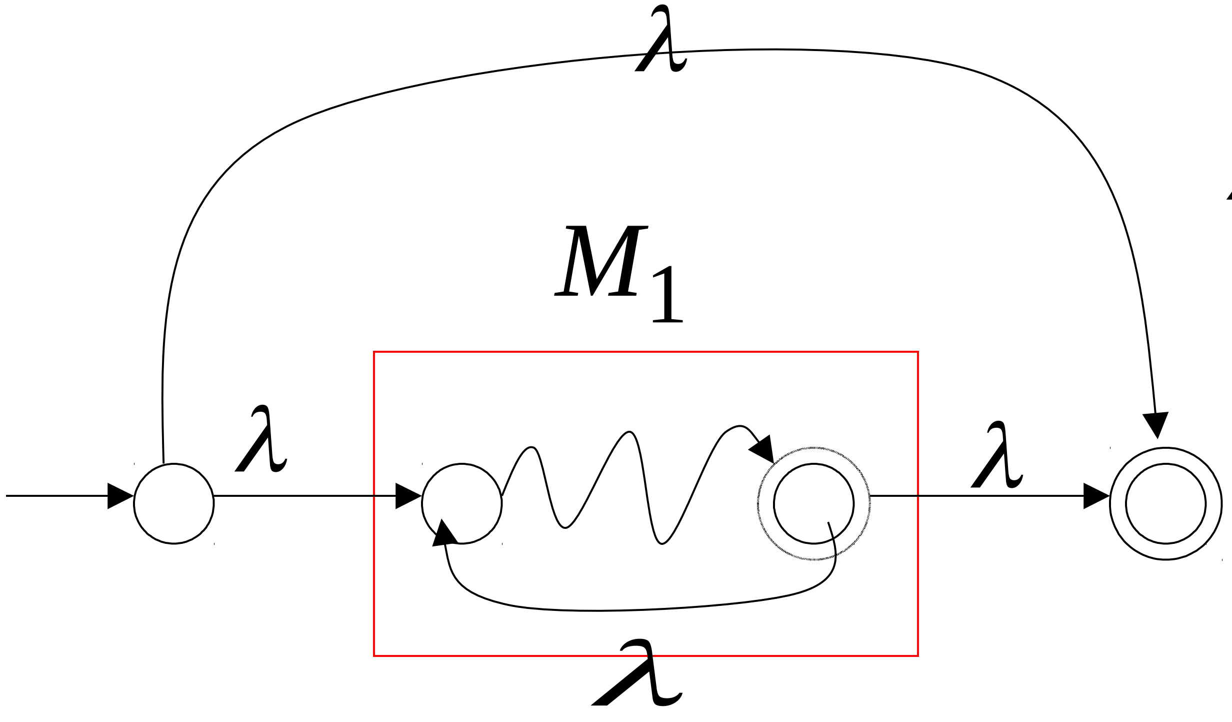
Example

NFA for $L_1 L_2 = \{a^n b\}\{ba\} = \{a^n bba\}$



Star Operation

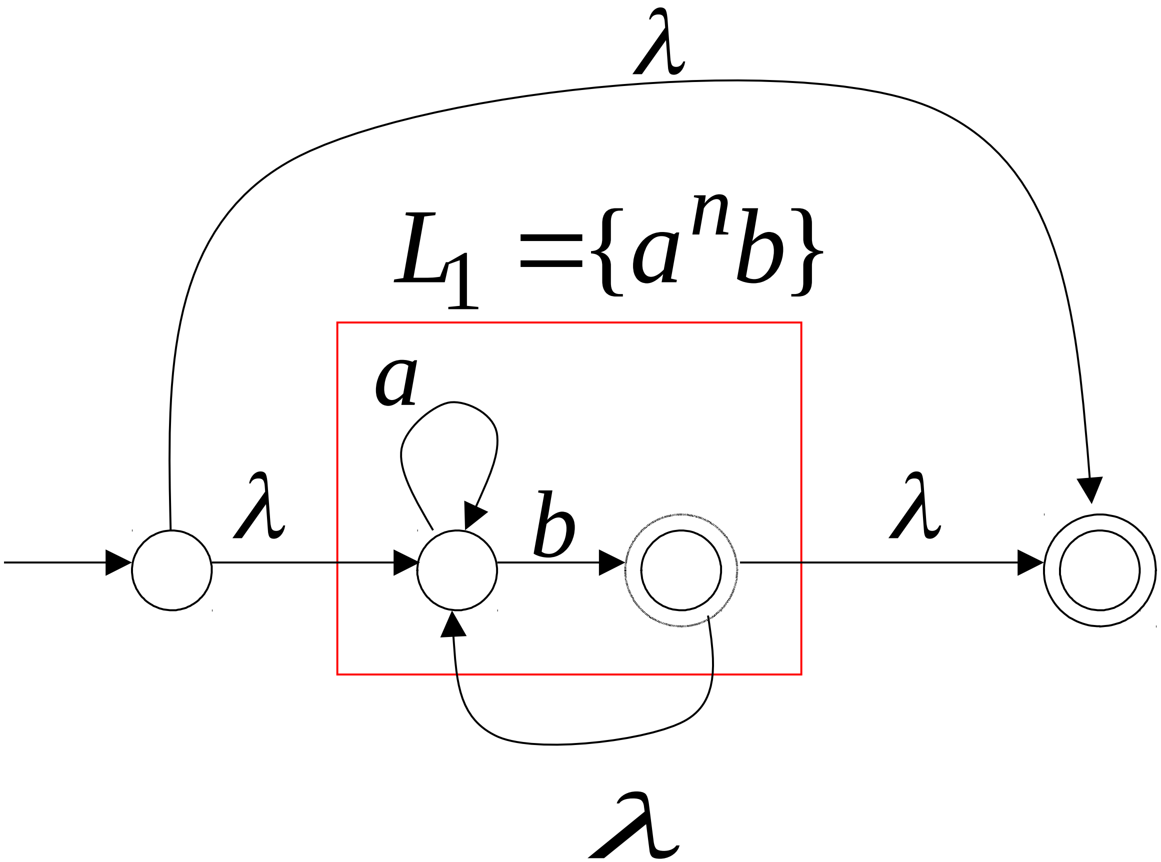
NFA for $L_1{}^*$



$\lambda \in L_1{}^*$

Example

NFA for $\quad L_1{}^* = \{a^n b\}^*$



Example

Regular expression $r = (0+1)^*\,00\,(0+1)^*$

$L(r)$ = { all strings with at least two consecutive 0 }

Example

Regular expression $r = (1 + 01)^*(0 + \lambda)$

$L(r)$ = { all strings without
two consecutive 0 }

Equivalent Regular Expressions

Definition:

Regular expressions r_1 and r_2

are **equivalent** if $L(r_1) = L(r_2)$

Theorem

$$\left\{\begin{array}{c}\text{Languages} \\ \text{Generated by} \\ \text{Regular Expressions}\end{array}\right\} = \left\{\begin{array}{c}\text{Regular} \\ \text{Languages}\end{array}\right\}$$

Proof - Part 1

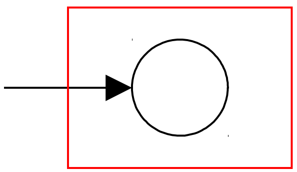
1. For any regular expression r
   the language $L(r)$ is regular

Proof by induction on the size of r

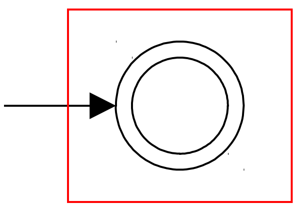
Induction Basis

Primitive Regular Expressions: $\emptyset, \quad \lambda, \quad \alpha$

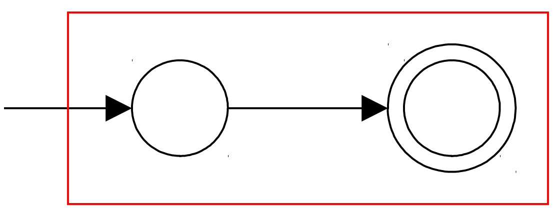
NFAs



$$L(M_1) = \emptyset = L(\emptyset)$$



$$L(M_2) = \{\lambda\} = L(\lambda)$$



$$L(M_3) = \{a\} = L(a)$$

regular languages

Inductive Hypothesis

Assume for regular expressions r_1 and r_2 that $L(r_1)$ and $L(r_2)$ are regular languages

Inductive Step

We will prove:

$$L(r_1 + r_2)$$

$$L(r_1 \cdot r_2)$$

$$L(r_1 *)$$

$$L((r_1))$$

Are regular Languages

By definition of regular expressions:

$$L(r_1 + r_2) = L(r_1) \cup L(r_2)$$

$$L(r_1 \cdot r_2) = L(r_1)\, L(r_2)$$

$$L(r_1 *) = (L(r_1))*$$

$$L((r_1)) = L(r_1)$$

By inductive hypothesis we know:

$L(r_1)$ and $L(r_2)$ are regular languages

We also know:

Regular languages are closed under

union $L(r_1) \cup L(r_2)$

concatenation $L(r_1)\, L(r_2)$

star $(L(r_1))^*$

Therefore:

$$\left.\begin{array}{l} L(r_1 + r_2) = L(r_1) \cup L(r_2) \\ \\ L(r_1 \cdot r_2) = L(r_1)\, L(r_2) \\ \\ L(r_1 *) = (L(r_1))* \end{array}\right\} \text{Are regular languages}$$

And trivially:

$L((r_1))$ is a regular language

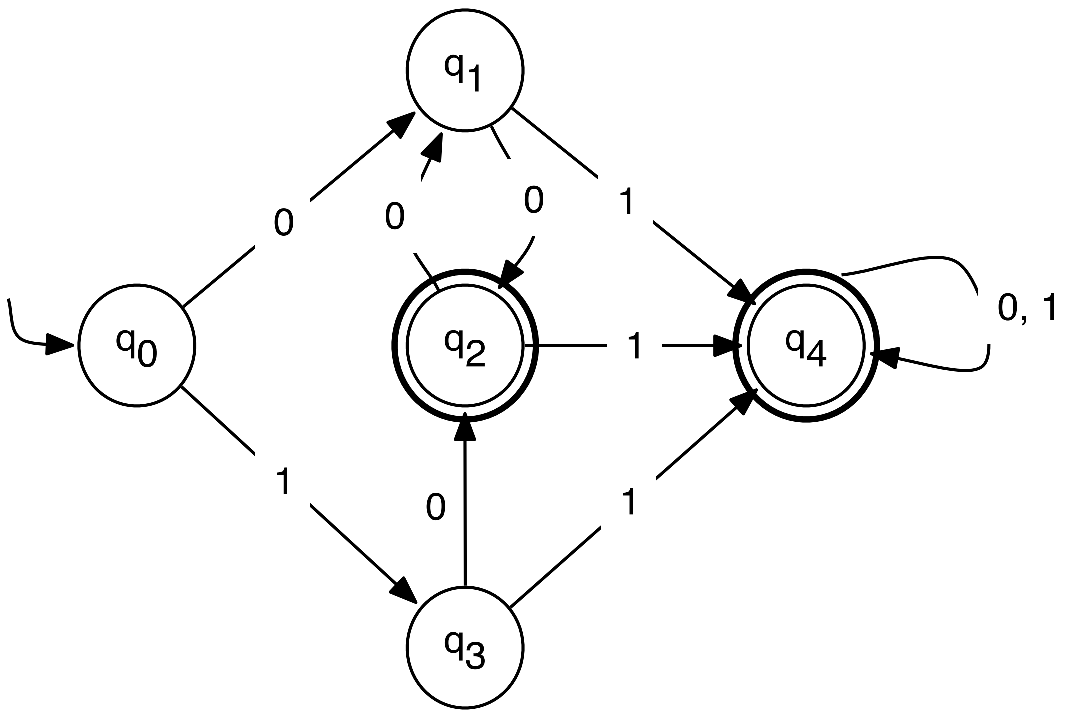
DFA minimization

- Steps:
  - Remove unreachable states first
  - Detect equivalent states
- Table-filing algorithm (checklist):
  - First, mark X for accept vs. non-accepting
  - Pass 1:
    - Then mark X where you can distinguish by just using one symbol transition
    - Also mark = whenever states are equivalent.
  - Pass 2:
    - Distinguish using already distinguished states (one symbol)
  - Pass 3:
    - Repeat for 2 symbols (on the state pairs left undistinguished)
    - …
  - Terminate when all entries have been filled
  - Finally modify the state diagram by keeping one representative state for every equivalent class

State Minimization Algorithm

- Mark all *final* states *distinguishable* from *non-final* states (strings of length 0 distinguish these states, obviously)
- *Repeat* until no *new* unmarked pairs are marked distinguishable:
  - For all *unmarked* pairs of states, (**p**,**q**):
    - For each letter, **c**, of the alphabet **Σ**:
      - If **δ**(**p**,**c**) and **δ**(**q**,**c**) are *distinguishable*, mark **p** and **q** *distinguishable*
- Combine each group of remaining *mutually indistinguishable* states into a *single state*

Example



Example

- Start by grouping *final* vs. *non-final* states:
  - $\{q_2, q_4\}$ vs. $\{q_0, q_1, q_3\}$
  - Mark all 6 pairings between these groups distinguishable:

	q_0	q_1	q_2	q_3	q_4
q_0			x		x
q_1			x		x
q_2				x	
q_3					x
q_4					

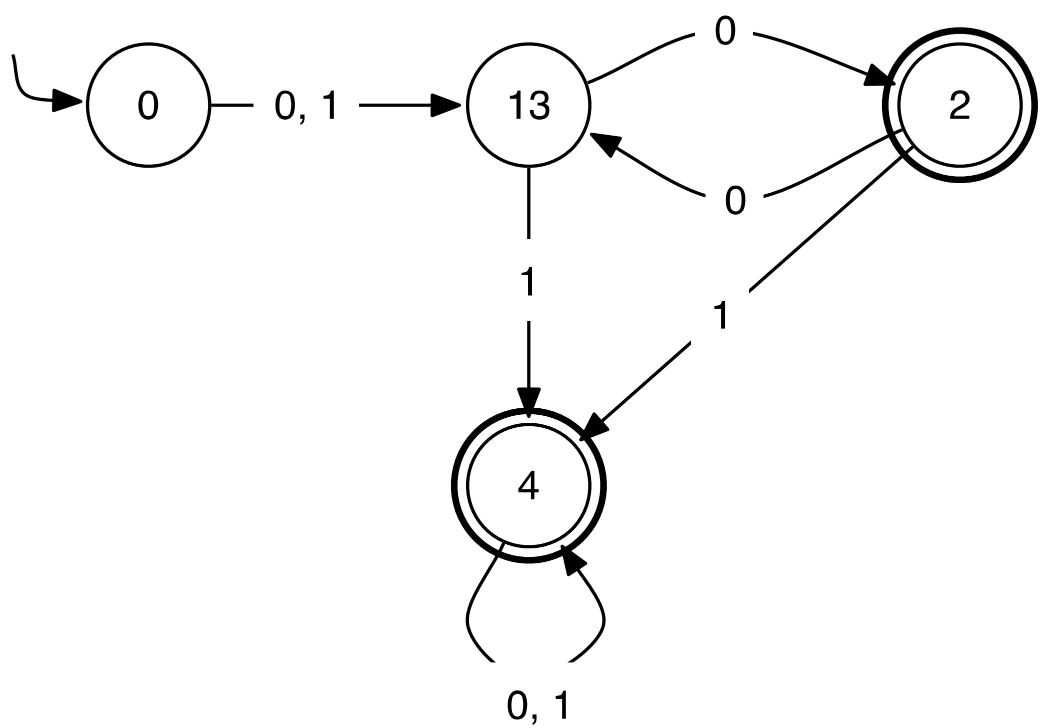
Example

- Check remaining unmarked pairs:
  - (q_2,q_4): $\delta(q_2,0) = q_1$, $\delta(q_4,0) = q_4$, => *distinguishable*
  - (q_0,q_1): $\delta(q_0,0) = q_1$, $\delta(q_1,0) = q_2$, => *distinguishable*
  - (q_0,q_3): $\delta(q_0,0) = q_1$, $\delta(q_3,0) = q_2$, => *distinguishable*
  - (q_1,q_3): $\delta(q_1,0) = \delta(q_3,0)$ *and* $\delta(q_1,1) = \delta(q_3,1)$, => *indistinguishable*

	q_0	q_1	q_2	q_3	q_4
q_0		x	x	x	x
q_1			x		x
q_2				x	x
q_3					x
q_4					

Result

Combine q_1 and q_3



Myhill Nerode Theorem:

The following three statements are equivalent

1. The set L ϵ $\sum^*$ is accepted by some FSA
2. L is the union of some of the equivalence classes of a right invariant equivalence relation of finite index.
3. Let equivalence relation R_L be defined by :

   xR_Ly iff for all z in $\sum^*$ xz is in L exactly when yz is in L.

   Then R_L is of finite index.

Theorem Proof:

There are three conditions:

Condition (i) implies condition (ii)

Condition (ii) implies condition (iii)

Condition (iii) implies condition (i)

Equivalence Relation

A binary relation $\sim$ over a set X is an equivalence relation if it satisfies

Reflexivity

Symmetry

Transitivity

Condition (i) implies condition (ii)

Proof:

Let L be a regular language accepted by a DFSA

$M = (Q,\sum,\delta,q_0,F)$.

Define R_M on $\sum^*$

$x \, R_M \, y$ if $\delta(q_0 , x) = \delta(q_0 , y)$

In order to show that its an equivalence relation it has to satisfy three properties.

$\delta(q_0 , x) = \delta(q_0 , x)$ --- Reflexive

If $\delta(q_0 , x) = \delta(q_0 , y)$ then

$\delta(q_0 , y) = \delta(q_0 , x)$ --- Symmetry

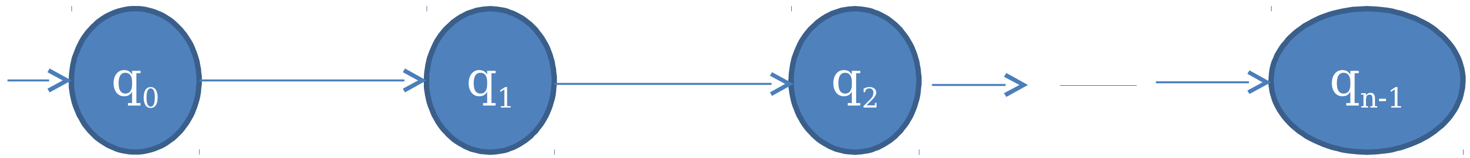
If $\delta(q_0 , x) = \delta(q_0 , y)$

$\delta(q_0 , y) = \delta(q_0 , z)$ then

$\delta(q_0 , x) = \delta(q_0 , z)$ --- Transitive

Index of an Equivalence relation:

There are N states



If This R_M is an Equivalence Relation, Then the index of R_M is at most the <u>number of States of M</u>

Right invariant

If $x \; R_M \; y$

Then $xz \; R_M \; yz$ for any $z \in \sum^*$

Then we say R_M is Right invariant

Proof:

$$\delta(q_0 , x) = \delta(q_0 , y)$$

$$\delta(q_0 , xz) = \delta(\delta(q_0 , x), z)$$

$$= \delta(\delta(q_0 , y), z)$$

$$= \delta(q_0 , yz)$$

Therefore R_M is right invariant

some

L is the union of ~~sum~~ of the equivalence classes of that relation.

If the Equivalence Relation R_M has n states.

$$S_0 , S_1 , S_2, \ldots\ldots, S_i , \ldots\ldots\ldots , S_{n-1}$$

$$| \quad | \quad | \quad | \quad |$$

$$q_0 , q_1 , q_2 ,\ldots\ldots, q_i ,\ldots\ldots\ldots, q_{n-1}$$

Condition (ii) implies condition (iii) :

Proof:

Let E be an equivalence relation as defined in (ii).

We have to prove that

E is a Refinement of R_L.

What is Refinement?

x E y | x,y є to same equivalence class of E

xz E yz | xz is related to yz for any z є ∑*

L is the union of ~~sum~~ some of the equivalence classes of E. If L contains this equivalence class then xz and yz are in L or it may not be in L.

Then we can say that

$x\ R_L\ y$

Hence it is proved that every equivalence class in E is an Equivalence class in R_L

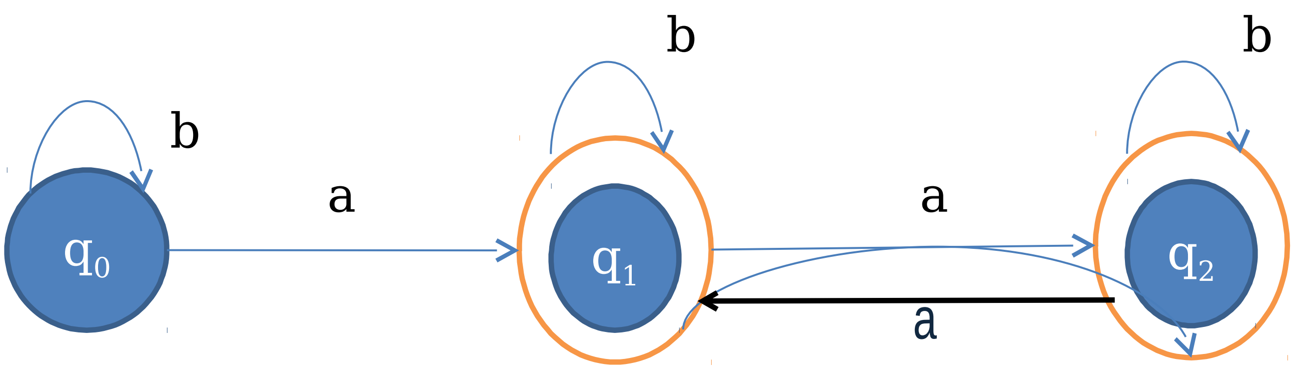
Then we can say that E is a Refinement of R_L

E is of finite index

Index of R_L <= index of E

therefore R_L is of Finite index.

Example : DFA



a

L ={ w | w contains a stings having atleast one a}

$\sum^*$ is partioned into three equivalence class J_0, J_1, J_2

J_0 – strings which do not contain an a

J_1 – strings which contain odd number of a’s

J_2 - strings which contain even number of a’s

$$L = J_1 \cup J_2$$

Condition (iii) implies condition (i)

Proof:

R_L is right invariant

$x \; R_L \; y$ if $xz \in L$ $yz \in L$

Therefore if $z = wz$ then

$xwz \in L$ $ywz \in L$ for any w and z

Then $xwz \; R_L \; ywz$

Hence R_L is Right invariant

Define an FSA $M' = (Q', \sum, \delta', q_0', F')$ as follows:

For each equivalence class of R_L, we have a state in Q'.

$|Q'|$ = index of R_L

If $x \in \sum^*$ denote the Equivalence class of R_L to which $x \in$ to [x]

$q_0' = [\epsilon]$ belongs to initial state / one equivalence class.

For symbol $a \in \sum$

$$\delta'([x],a) = [xa]$$

This definition is consistent because R_L is right invariant.

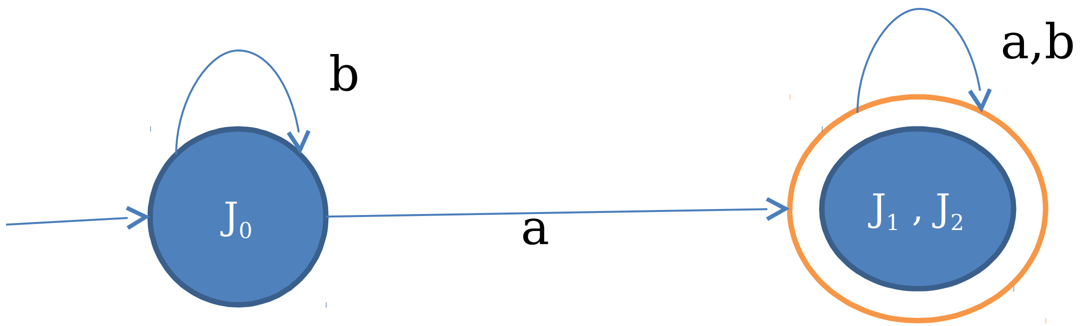
If $xR_L y$ then

$$\delta([x],a) = [ya]$$

Because x,y belong to same class and Right invariant.

Therefore we can say that L is accepted by a FSA.

J_0 and $J_1 \cup J_2$ are the two equivalence classes in R_L



To show that a given language is not Regular:

$L = \{a^n b^n \mid n >= 1\}$

Assume that L is Regular

Then by Myhill Nerode theorem we can say that L is the union of some ~~sum~~ of the Equivalence classes and etc

a, aa,aaa,aaaa,……..

Each of this cannot be in different equivalence classes.

$a^n \sim a^m$ for $m \neq n$

By Right invariance

$a^n b^n \sim a^m b^n$ for $m \neq n$

Hence contradiction

The L cannot be regular.

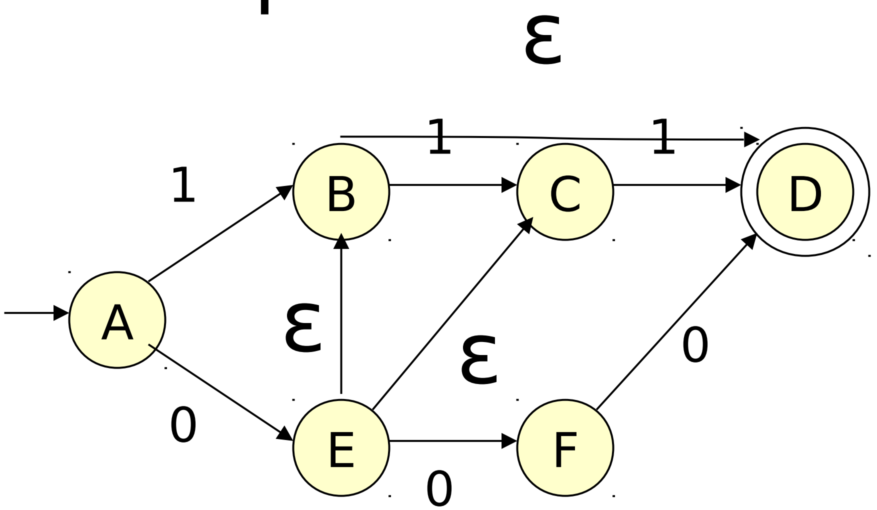
NFA's With ε-Transitions

We can allow state-to-state transitions on ε input.

These transitions are done spontaneously, without looking at the input string.

A convenience at times, but still only regular languages are accepted.

Example: ε-NFA



		0	1	ε
→	A	{E}	{B}	∅
	B	∅	{C}	{D}
	C	∅	{D}	∅
	D	∅	∅	∅
*	E	{F}	∅	{B, C}
	F	{D}	∅	∅

Closure of States

CL(q) = set of states you can reach from state q following only arcs labeled ε.

$$CL(q) = \hat{\delta}(q, \varepsilon)$$

Example: CL(A) = {A};

CL(E) = {B, C, D, E}.

Closure of a set of states = union of the closure of each state.

Language of an ε-NFA is the set of strings w such that $\hat{\delta}(q_0, w)$ contains a final state.

Equivalence of NFA, ε-NFA

Every NFA is an ε-NFA.
It just has no transitions on ε.

Take an ε-NFA and construct an NFA that accepts the same language.

We do so by combining ε–transitions with the next transition on a real input.

Start with an ε-NFA with states Q, inputs Σ, start state q_0, final states F, and transition function δ_E.

Construct an "ordinary" NFA with states Q, inputs Σ, start state q_0, final states F', and transition function δ_N.

Compute $\delta_N(q, a)$ as follows:
Let S = CL(q).

$\delta_N(q, a)$ is the union over all p in S of $\delta_E(p, a)$.

ie. $\delta_N(q_0, a) = CL(\delta_E(q_0, a))$

F' = the set of states q such that CL(q) contains a state of F.

Thank you!