THEORY OF COMPUTATION

UNIT I:

Finite State Machines

By:
Priyamvada S Chauhan

Syllabus

Module No.	Unit No.	Topics	Theory Hrs.
1.0		**Basic Concepts and Finite Automata**	**09**
	1.1	Importance of TCS, Alphabets, Strings, Languages, Closure properties, Finite Automata (FA) and Finite State machine (FSM).	
	1.2	Deterministic Finite Automata (DFA) and Nondeterministic Finite Automata (NFA): Definitions, transition diagrams and Language recognizers, Equivalence between NFA with and without ε- transitions, NFA to DFA Conversion, Minimization of DFA, FSM with output: Moore and Mealy machines, Applications and limitations of FA.	
2.0		**Regular Expressions and Languages**	**07**
	2.1	Regular Expression (RE),Equivalence of RE and FA, Arden's Theorem, RE Applications	
	2.2	Regular Language (RL), Closure properties of RLs, Decision properties of RLs, Pumping lemma for RLs.	
3.0		**Grammars**	**08**
	3.1	Grammars and Chomsky hierarchy	
	3.2	Regular Grammar (RG), Equivalence of Left and Right linear grammar, Equivalence of RG and FA.	

Syllabus

	3.3	**Context Free Grammars (CFG)** Definition, Sentential forms, Leftmost and Rightmost derivations, Parse tree, Ambiguity, Simplification and Applications, Normal Forms: Chomsky Normal Forms (CNF) and Greibach Normal Forms (GNF), Context Free language (CFL) - Pumping lemma, Closure properties.	
4.0		**Pushdown Automata(PDA)**	**04**
	4.1	Definition, Language of PDA,PDA as generator, decider and acceptor of CFG, Deterministic PDA , Non-Deterministic PDA, Application of PDA.	
5.0		**Turing Machine (TM)**	**09**
	5.1	Definition, Design of TM as generator, decider and acceptor, Variants of TM: Multitrack, Multitape, Universal TM, Applications, Power and Limitations of TMs.	
6.0		**Undecidability**	**02**
	6.1	Decidability and Undecidability, Recursive and Recursively Enumerable Languages, Halting Problem, Rice's Theorem, Post Correspondence Problem.	
		Total	**39**

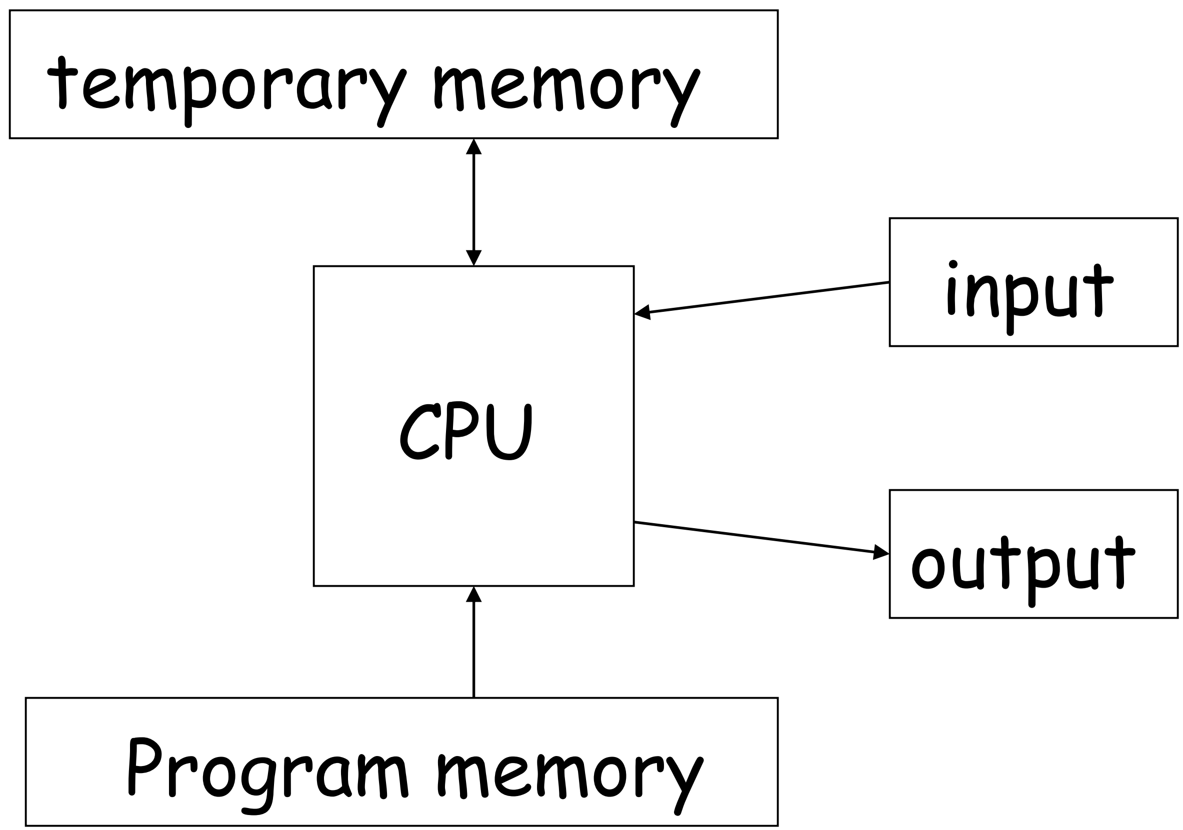
Learning/Course Outcomes

After completing this course, students will be able to:

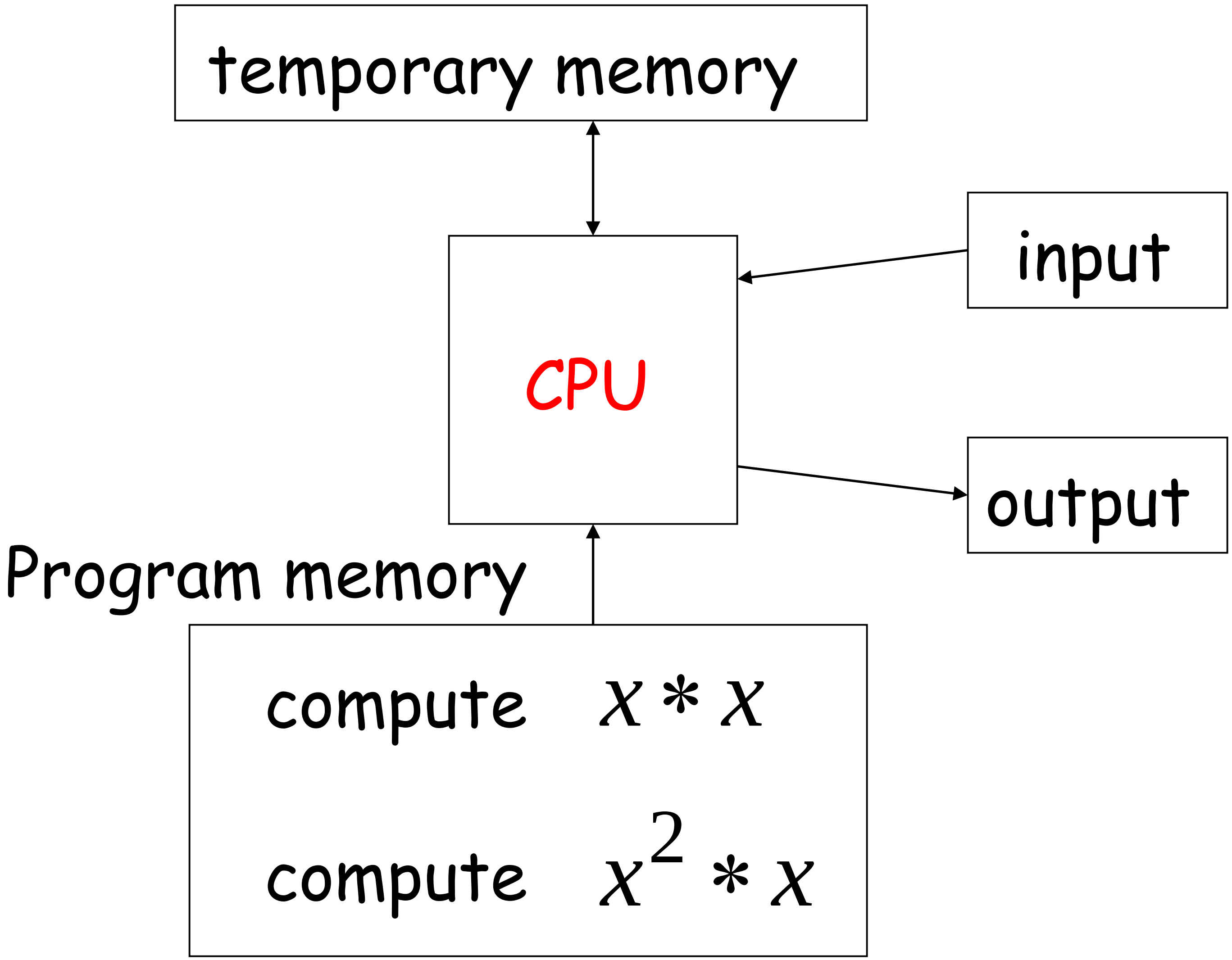
1. Illustrate the fundamentals of Theoretical Computer Science.
2. Construct Computational models including Finite Machine, Push Down Automata And Turing Machine.
3. Construct regular grammar for language and use several techniques for simplification.
4. Apply formal mathematical method to prove the properties of Formal Language.
5. Proof that certain languages are undecidable.

Introduction

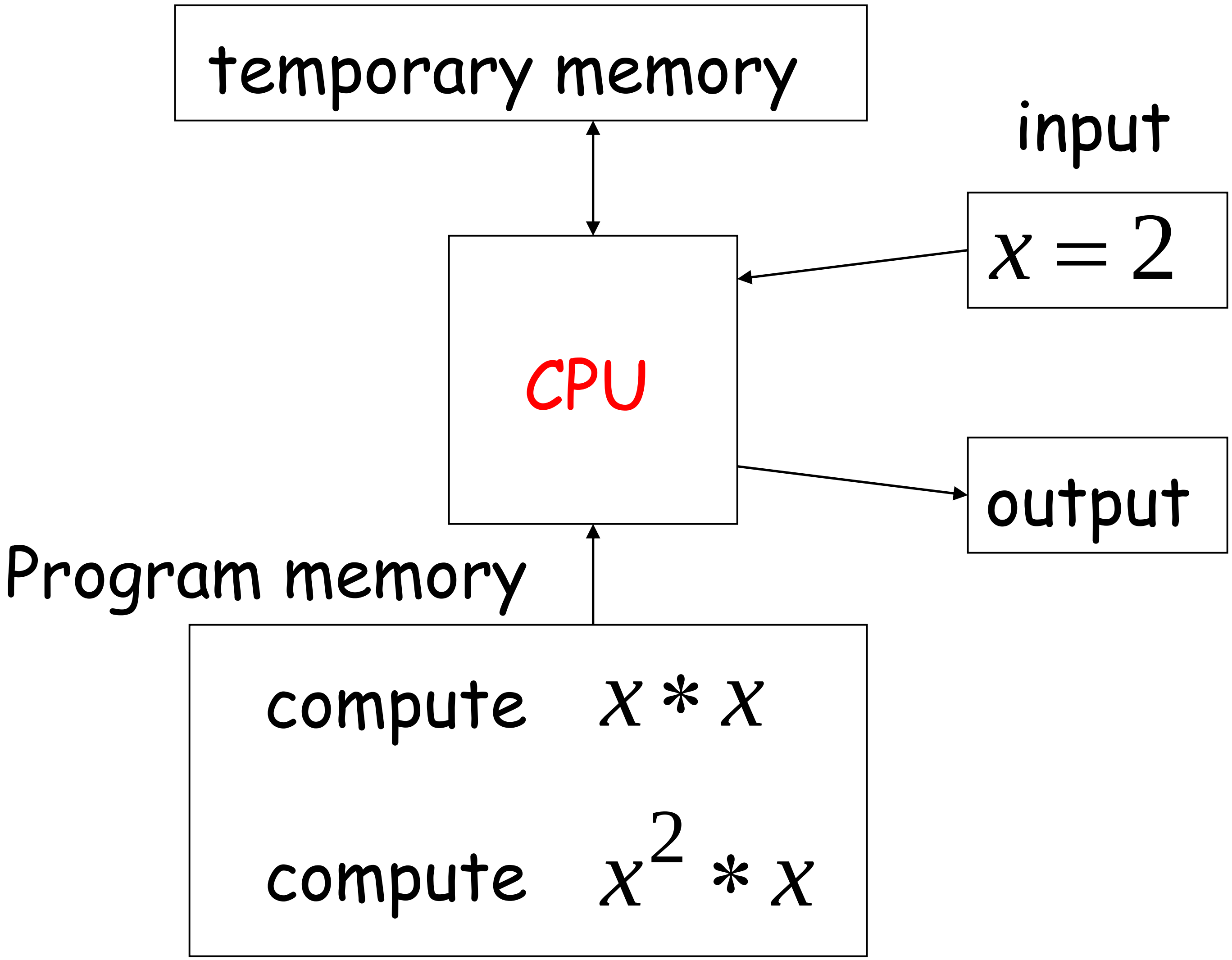
- Computations are designed to solve problems.
- Computational Devices
- Resources use by Computational Devices
- Programs are descriptions of computations written for execution on computers.
- The field of computer science is concerned with the development of methodologies for designing programs, and with the development of computers for executing programs.
- It is therefore of central importance for those involved in the field that the characteristics of programs, computers, problems, and computation be fully understood.


temporary memory
input
CPU
output
Program memory

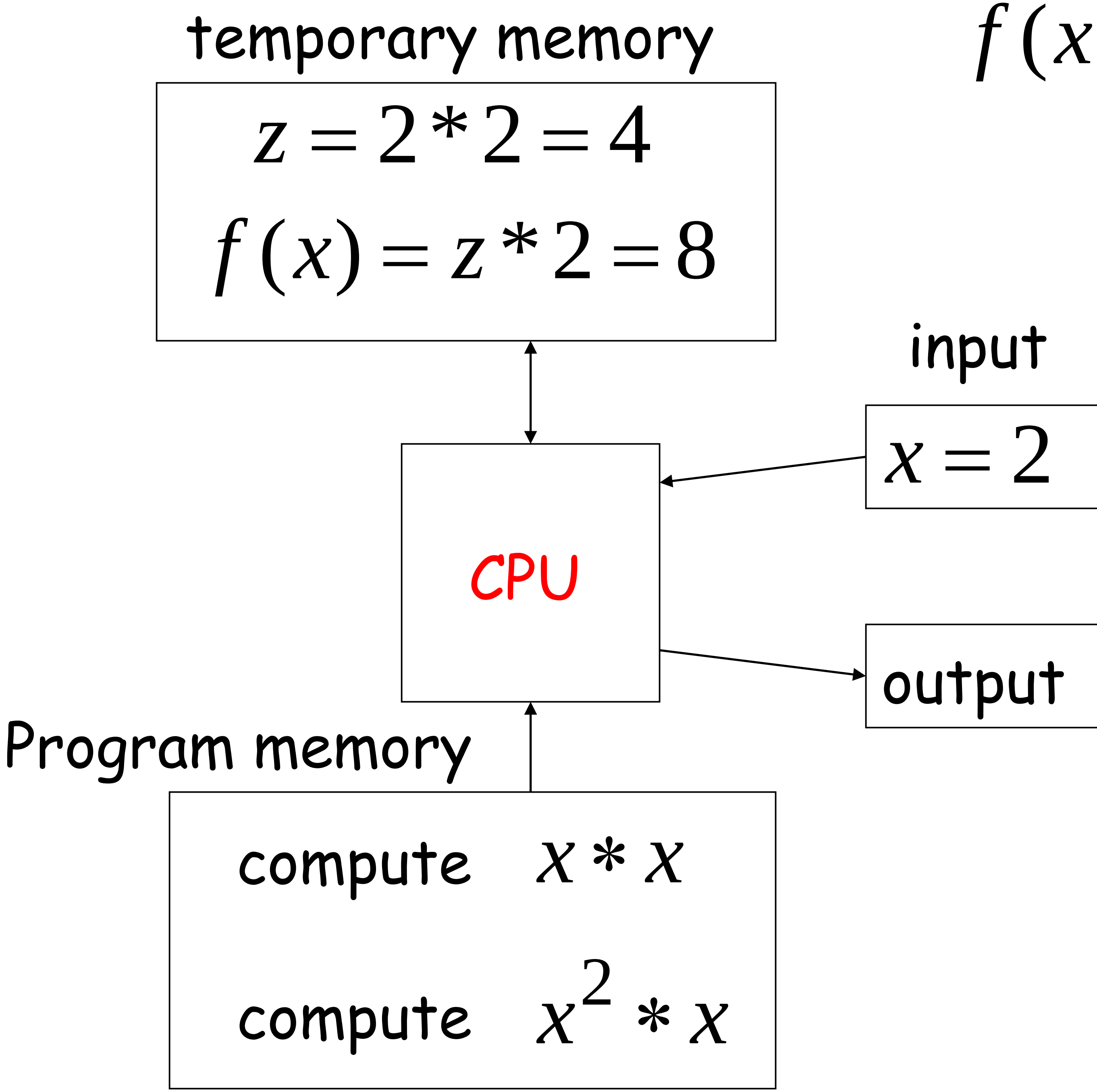
Example: $f(x)=x^3$



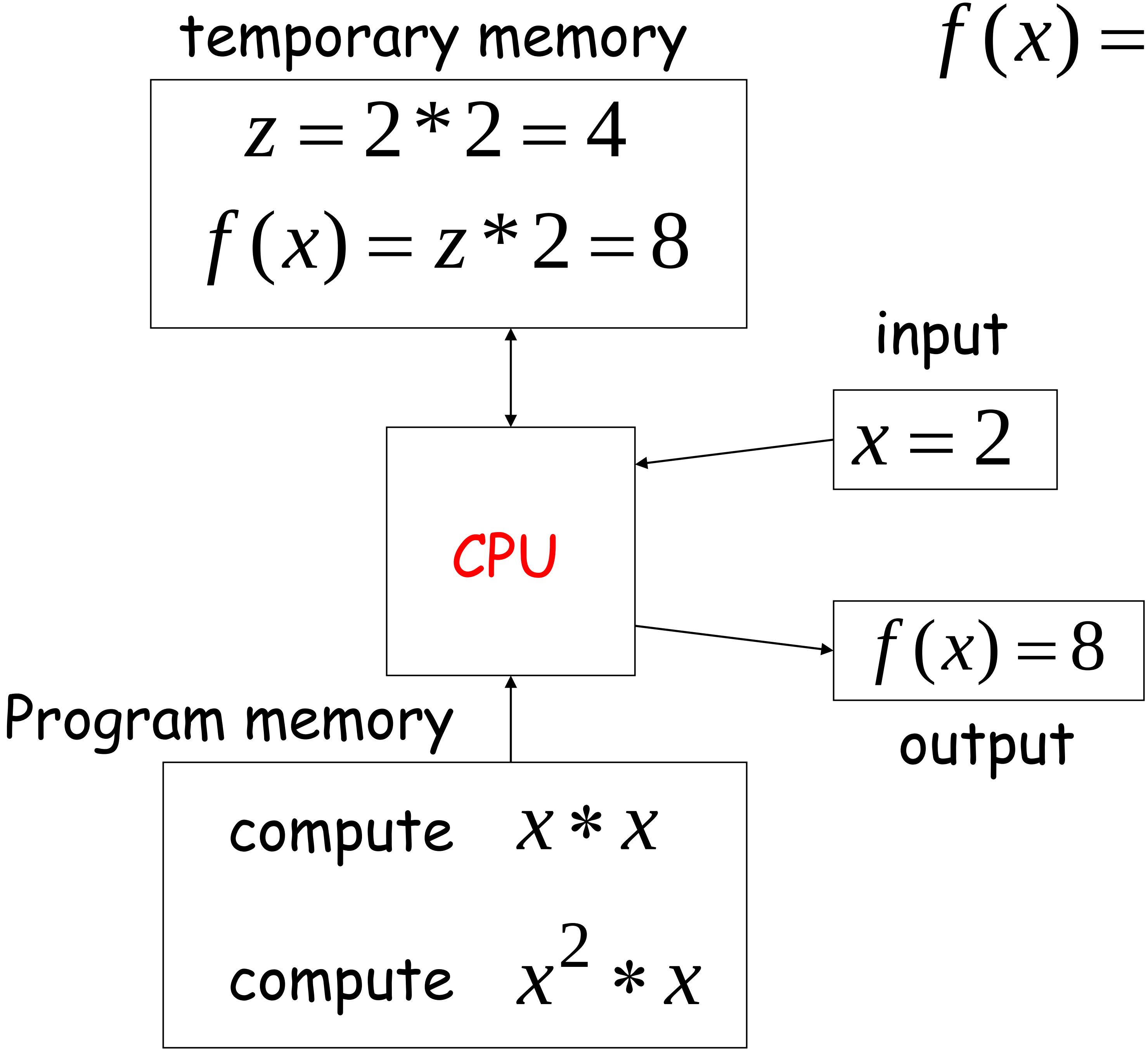
$$f(x) = x^3$$


temporary memory
input
$x = 2$
CPU
output
Program memory
compute $x * x$
compute $x^2 * x$

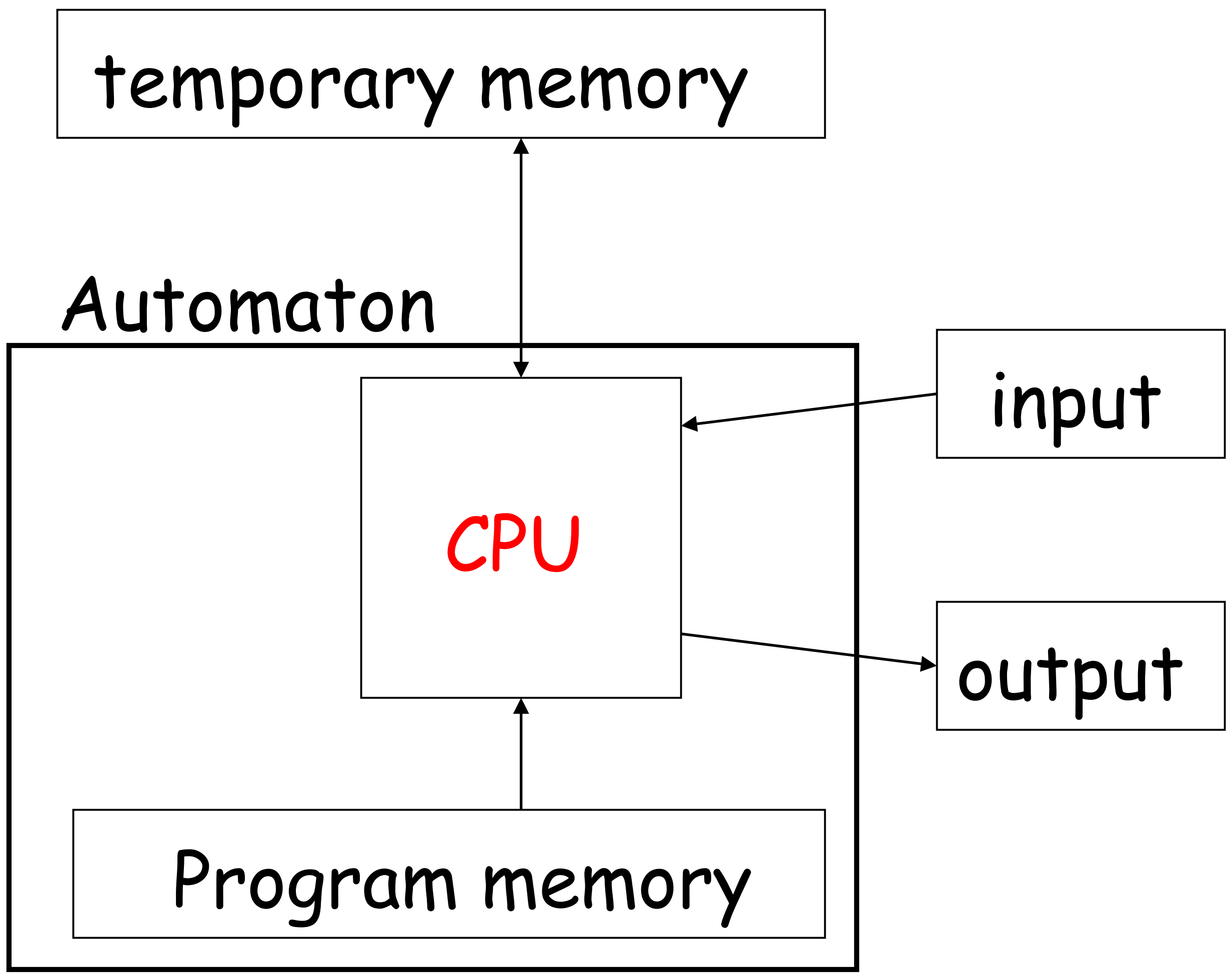
$$f(x) = x^3$$


temporary memory
$z = 2*2 = 4$
$f(x) = z*2 = 8$
input
$x = 2$
CPU
output
Program memory
compute $x*x$
compute x^2*x

$$f(x) = x^3$$


temporary memory
$z = 2*2 = 4$
$f(x) = z*2 = 8$
input
$x = 2$
CPU
$f(x) = 8$
output
Program memory
compute $x*x$
compute x^2*x

Automaton



temporary memory
Automaton
CPU
input
output
Program memory

Basic Building Blocks

- Symbol

  a, b, c…A, B, C…0,1,2…..

- Alphabet

  $\Sigma=\{a, b\}$ $\Sigma=\{0, 1\}$

- String

  aa, ab, bab, bba…….00,01,10,11,101

- Language

Language

Σ={a, b}

L1=set of all string of length 2

={aa, ab, ba, bb}

L2=set of all string of length 3

={aaa, aab, aba, abb, baa, bab, baa, bbb}

L3=set of all string where each string starts with 'a'

={a, aa, ab, aaa, aab, aba, abb, ……..}

- Language is finite

$\sum$ ={a, b} L1=set of all string of length 2

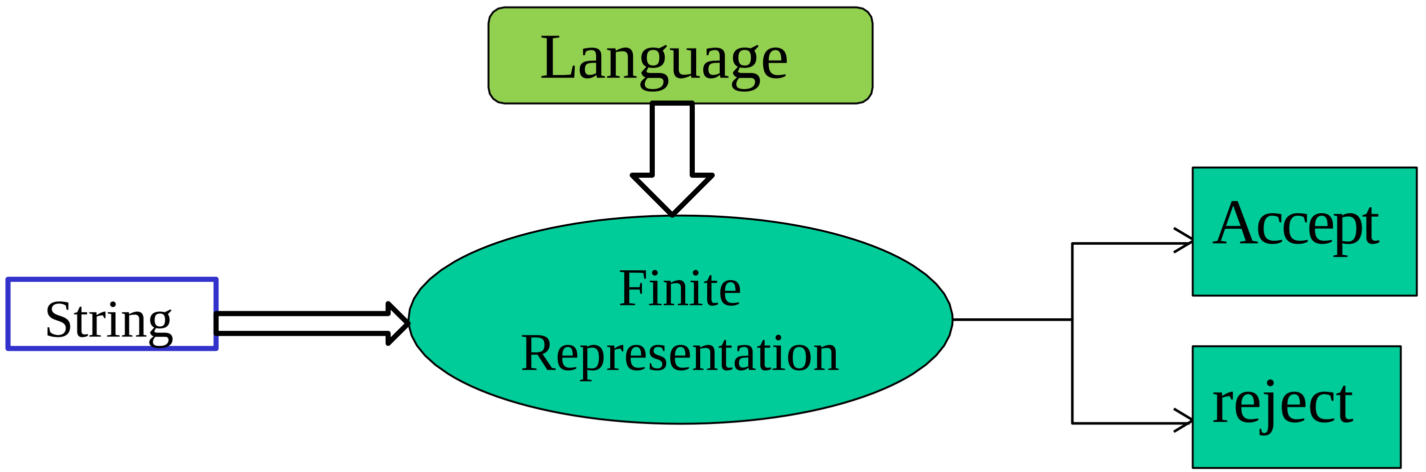
$L1 =$ {aa, ab, ba, bb}

- Language is infinite

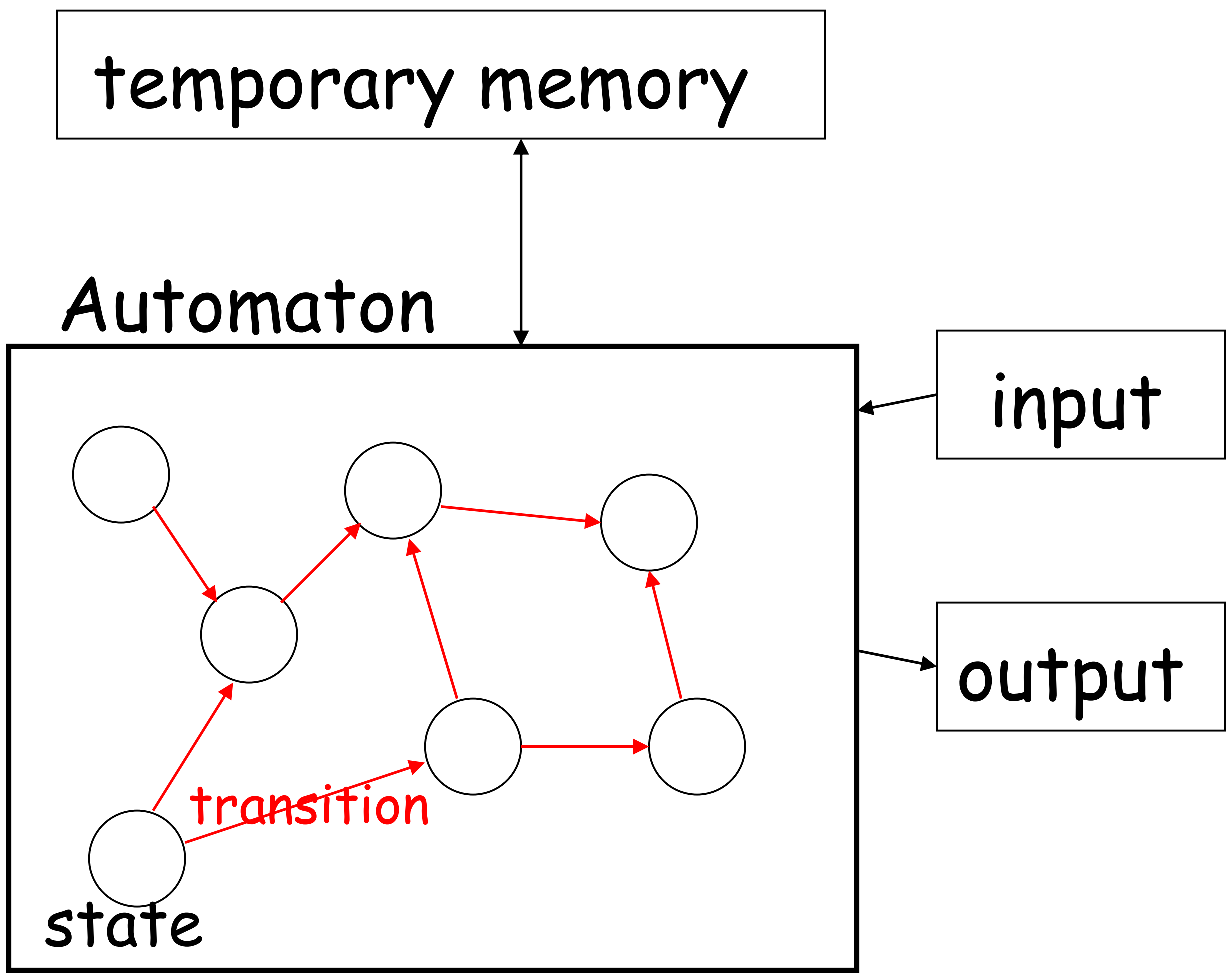
$\sum =$ {a, b} L3=set of all string where each string starts with ‘a

L2 $=$ {a, aa, ab, abb, aab, aba,..........}

- String
- Linear searching have limitations so…..

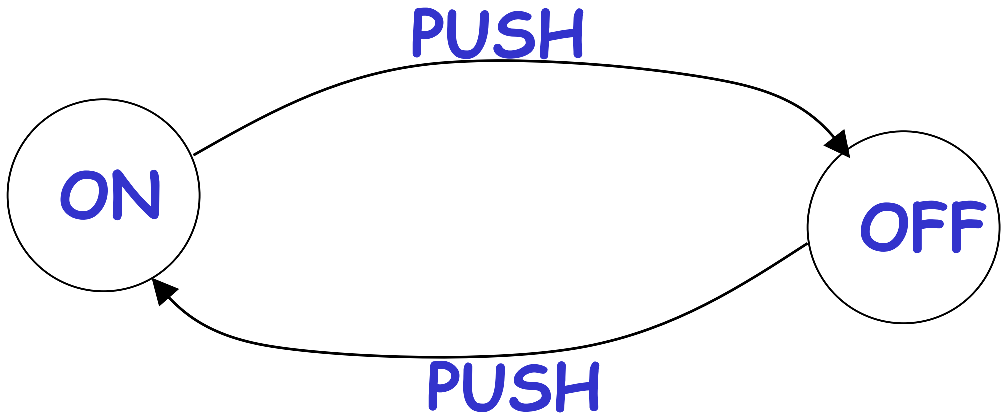

Language
String
Finite
Representation
Accept
reject

Automaton

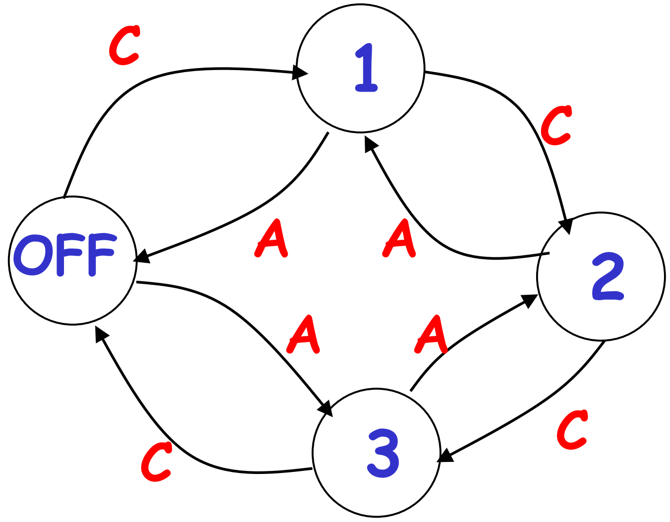


temporary memory
Automaton
input
output
transition
state

• Electric Switch



• FAN REGULATOR



Formal Definition

Deterministic Finite Automaton (DFA)

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

Q : set of states

Σ : input alphabet $\quad \lambda \notin \Sigma$

δ : transition function

q_0 : initial state

F : set of accepting states

Languages

Language: a set of strings

String: a sequence of symbols
from some alphabet

Example:

Strings: cat, dog, house

Language: {cat, dog, house}

Alphabet: $\Sigma = \{a, b, c, \ldots, z\}$

Languages are used to describe computation problems:

$$PRIMES = \{2,3,5,7,11,13,17,\ldots\}$$

$$EVEN = \{0,2,4,6,\ldots\}$$

Alphabet: $\Sigma = \{0,1,2,\ldots,9\}$

Alphabets and Strings

An alphabet is a set of symbols

Example Alphabet: $\Sigma = \{a, b\}$

A string is a sequence of symbols from the alphabet

Example Strings

a

ab

$abba$

$aaabbbaaba b$

$u = ab$

$v = bbbaaa$

$w = abba$

Decimal numbers alphabet $\Sigma = \{0,1,2,\ldots,9\}$

102345 567463386

Binary numbers alphabet $\Sigma = \{0,1\}$

100010001 101101111

Unary numbers alphabet $\Sigma = \{1\}$

Unary number:	1	11	111	1111	11111
Decimal number:	1	2	3	4	5

String Operations

$$w = a_1 a_2 \cdots a_n$$ $abba$

$$v = b_1 b_2 \cdots b_m$$ $bbbaaa$

Concatenation

$$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m$$ $abbabbbaaa$

$$w = a_1 a_2 \cdots a_n \qquad ababaaabbb$$

Reverse

$$w^R = a_n \cdots a_2 a_1 \qquad bbbaaababa$$

String Length

$$w = a_1 a_2 \cdots a_n$$

Length: $|w| = n$

Examples:

$$|abba| = 4$$

$$|aa| = 2$$

$$|a| = 1$$

Length of Concatenation

$$|uv| = |u| + |v|$$

Example:
$$u = aab, \quad |u| = 3$$
$$v = abaab, \quad |v| = 5$$

$$|uv| = |aababaab| = 8$$
$$|uv| = |u| + |v| = 3 + 5 = 8$$

Empty String

A string with no letters is denoted: λ **or** ε

Observations: $|\lambda| = 0$

$$\lambda w = w\lambda = w$$

$$\lambda abba = abba\lambda = ab\lambda ba = abba$$

Substring

Substring of string:

a subsequence of consecutive characters

String	Substring
$\underline{ab}bab$	ab
$\underline{abba}b$	$abba$
$ab\underline{b}ab$	b
$a\underline{bbab}$	$bbab$

Prefix and Suffix

$abbab$

Prefixes	Suffixes
λ	$abbab$
a	$bbab$
ab	bab
abb	ab
$abba$	b
$abbab$	λ

$$w = uv$$

prefix

suffix

Another Operation

$$w^n = \underbrace{ww\cdots w}_{n}$$

Example: $$(abba)^2 = abbaabba$$

Definition: $$w^0 = \lambda$$

$$(abba)^0 = \lambda$$

The * Operation

Σ^*: the set of all possible strings from alphabet Σ

$$\Sigma = \{a, b\}$$

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$$

The + Operation

Σ^+ : the set of all possible strings from alphabet Σ except λ

$$\Sigma = \{a, b\}$$

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$$

$$\Sigma^+ = \Sigma^* - \lambda$$

$$\Sigma^+ = \{a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$$

Review

- Informal Introduction to Finite Automata
- Symbol
- Alphabet
- String
- String Operations
- * (Kleene closure) and + (positive closure)

Languages

A language over alphabet Σ

is any subset of Σ^*

Examples:

$$\Sigma = \{a, b\}$$

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, \ldots\}$$

Language: $\{\lambda\}$

Language: $\{a, aa, aab\}$

Language: $\{\lambda, abba, baba, aa, ab, aaaaaa\}$

More Language Examples

Alphabet $\Sigma = \{a, b\}$

An infinite language $L = \{a^n b^n : n \geq 0\}$

$$\left.\begin{array}{l} \lambda \\ ab \\ aabb \\ aaaaabbbbb \end{array}\right\} \in L$$

$$abb \notin L$$

Prime numbers

Alphabet $\Sigma = \{0,1,2,\ldots,9\}$

Language:

$$PRIMES = \{x : x \in \Sigma^* \text{ and } x \text{ is prime}\}$$

$$PRIMES = \{2,3,5,7,11,13,17,\ldots\}$$

Even and odd numbers

Alphabet $\Sigma = \{0,1,2,\ldots,9\}$

$$EVEN = \{x : x \in \Sigma^* \text{ and } x \text{ is even}\}$$

$$EVEN = \{0,2,4,6,\ldots\}$$

$$ODD = \{x : x \in \Sigma^* \text{ and } x \text{ is odd}\}$$

$$ODD = \{1,3,5,7,\ldots\}$$

Unary Addition

Alphabet: $\Sigma = \{1, +, =\}$

Language:

$$ADDITION = \{x + y = z \ : x = 1^n, y = 1^m, z = 1^k, \\ n + m = k\}$$

$$11 + 111 = 11111 \in ADDITION$$

$$111 + 111 = 111 \notin ADDITION$$

Note that:

Sets $\emptyset = \{ \} \neq \{\lambda\}$

Set size $|\{ \}| = |\emptyset| = 0$

Set size $|\{\lambda\}| = 1$

String length $|\lambda| = 0$

Operations on Languages

The usual set operations

$$\{a, ab, aaaa\} \cup \{bb, ab\} = \{a, ab, bb, aaaa\}$$

$$\{a, ab, aaaa\} \cap \{bb, ab\} = \{ab\}$$

$$\{a, ab, aaaa\} - \{bb, ab\} = \{a, aaaa\}$$

Complement: $\overline{L} = \Sigma^* - L$

$$\overline{\{a, ba\}} = \{\lambda, b, aa, ab, bb, aaa, \ldots\}$$

Reverse

Definition: $L^R = \{w^R : w \in L\}$

Examples: $\{ab, aab, baba\}^R = \{ba, baa, abab\}$

$$L = \{a^n b^n : n \geq 0\}$$

$$L^R = \{b^n a^n : n \geq 0\}$$

Concatenation

Definition: $$L_1 L_2 = \{xy : x \in L_1, y \in L_2\}$$

Example: $$\{a, ab, ba\}\{b, aa\}$$

$$= \{ab, aaa, abb, abaa, bab, baaa\}$$

Another Operation

Definition: $$L^n = \underbrace{LL \cdots L}_{n}$$

$$\{a,b\}^3 = \{a,b\}\{a,b\}\{a,b\} =$$
$$\{aaa, aab, aba, abb, baa, bab, bba, bbb\}$$

Special case: $L^0 = \{\lambda\}$

$$\{a, bba, aaa\}^0 = \{\lambda\}$$

Star-Closure (Kleene *)

All strings that can be constructed from L

Definition: $L^* = L^0 \cup L^1 \cup L^2 \cdots$

Example:

$$\{a, bb\}^* = \left\{ \begin{array}{l} \lambda, \\ a, bb, \\ aa, abb, bba, bbbb, \\ aaa, aabb, abba, abbbb, \ldots \end{array} \right\}$$

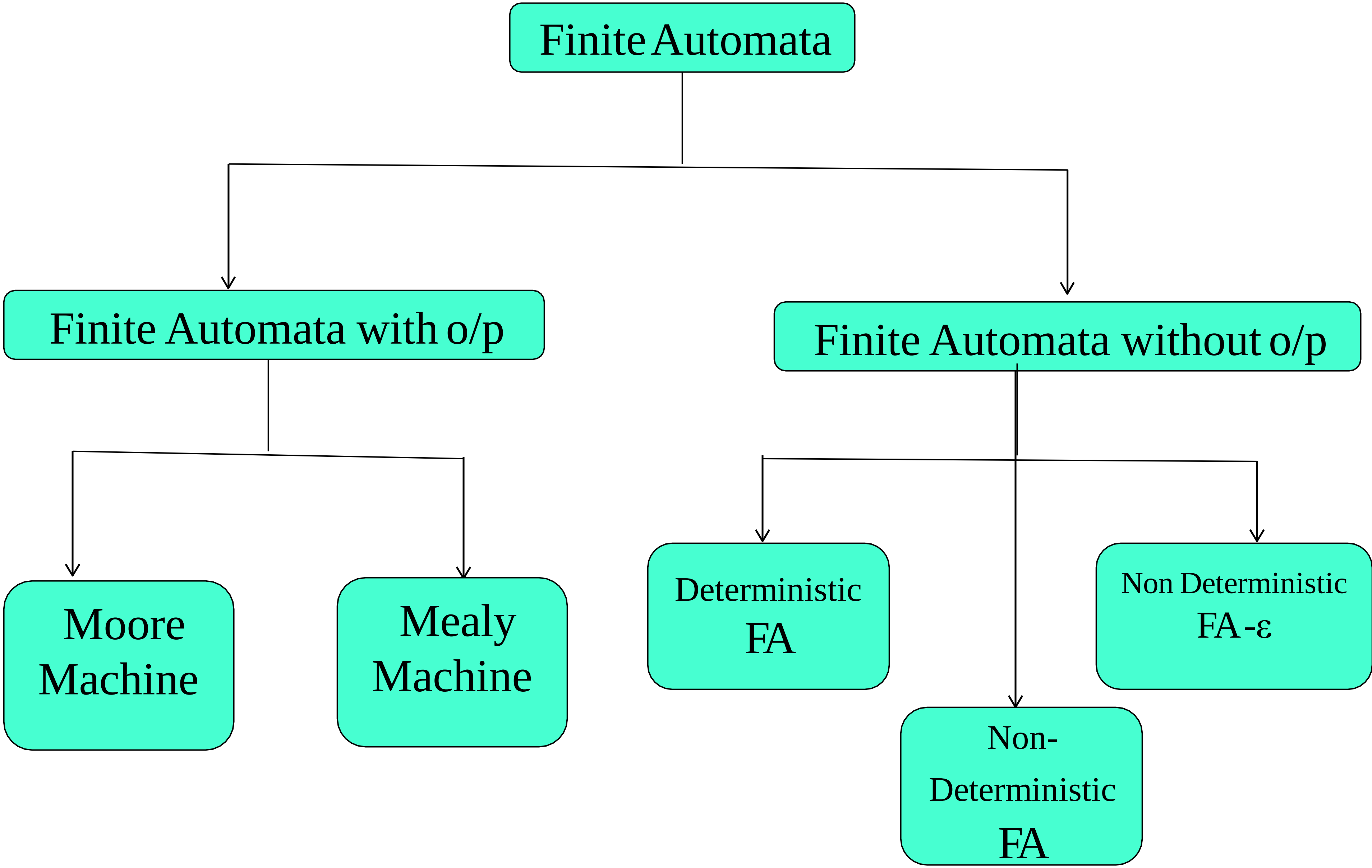
Positive Closure

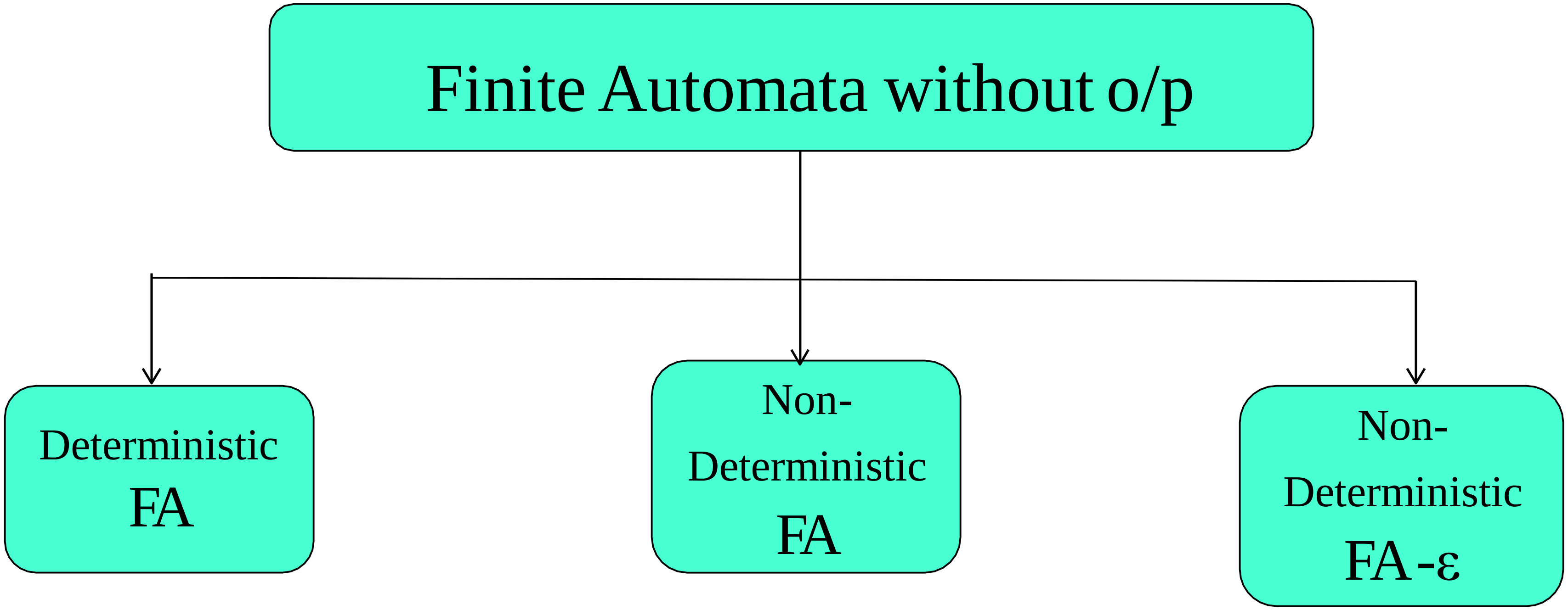
Definition: $L^{+} = L^{1} \cup L^{2} \cup \cdots$

Same with L^{*} but without the λ

$$\{a, bb\}^{+} = \left\{\begin{array}{l} a, bb, \\ aa, abb, bba, bbbb, \\ aaa, aabb, abba, abbbb, \ldots \end{array}\right\}$$

Finite Automaton/ Finite State Machine


Finite Automata
Finite Automata with o/p
Finite Automata without o/p
Moore Machine
Mealy Machine
Deterministic FA
Non-Deterministic FA
Non Deterministic FA -ε


Finite Automata without o/p
Deterministic FA
Non-Deterministic FA
Non-Deterministic FA -ε

Formal Definition

Deterministic Finite Automaton (DFA)

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

Q : set of states

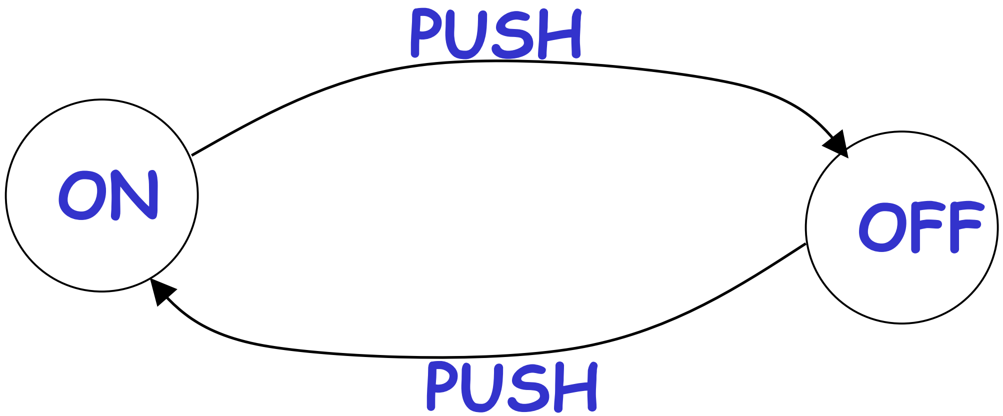
Σ : input alphabet $\qquad \lambda \notin \Sigma$

δ : transition function

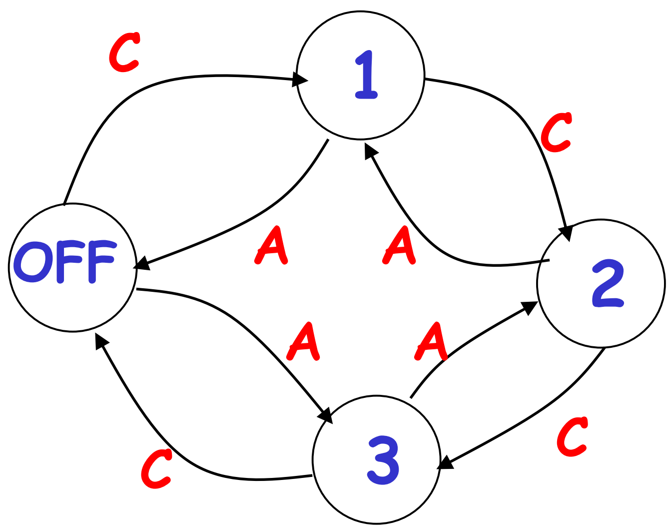
q_0 : initial state

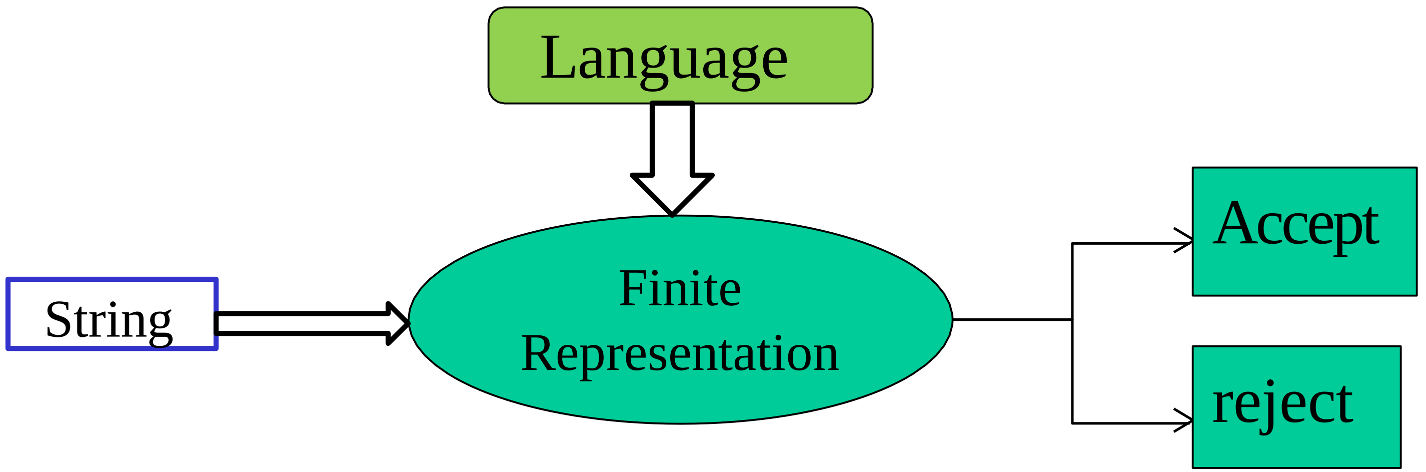
F : set of accepting states

- Electric Switch

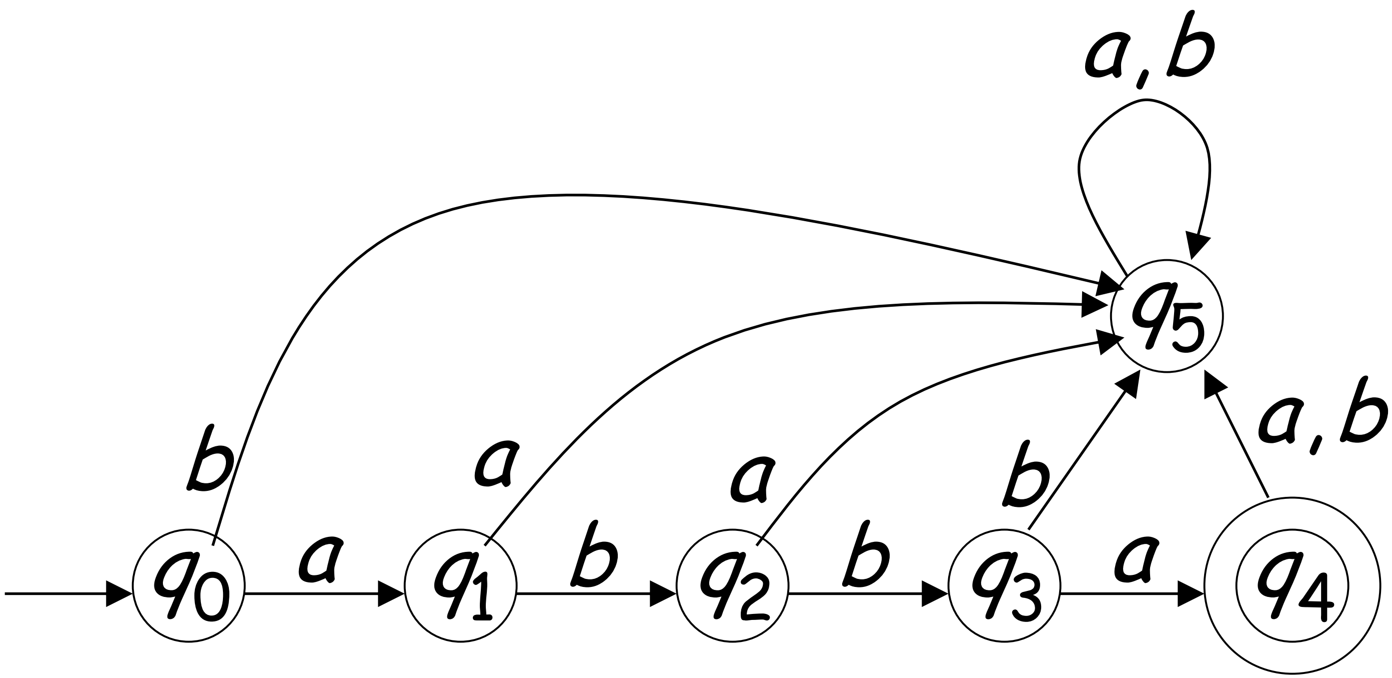


- FAN REGULATOR




Language
String
Finite Representation
Accept
reject

• DFA

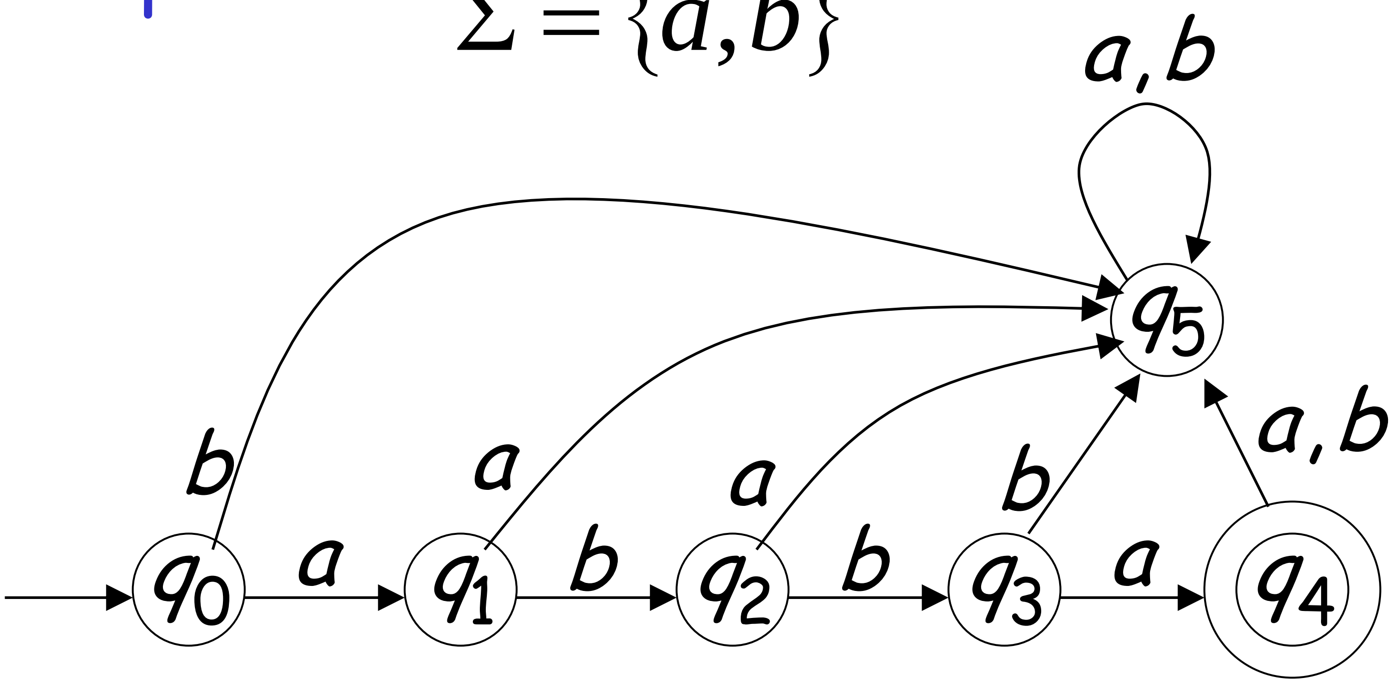


Input Alphabet Σ

$\lambda \notin \Sigma$:the input alphabet never contains λ

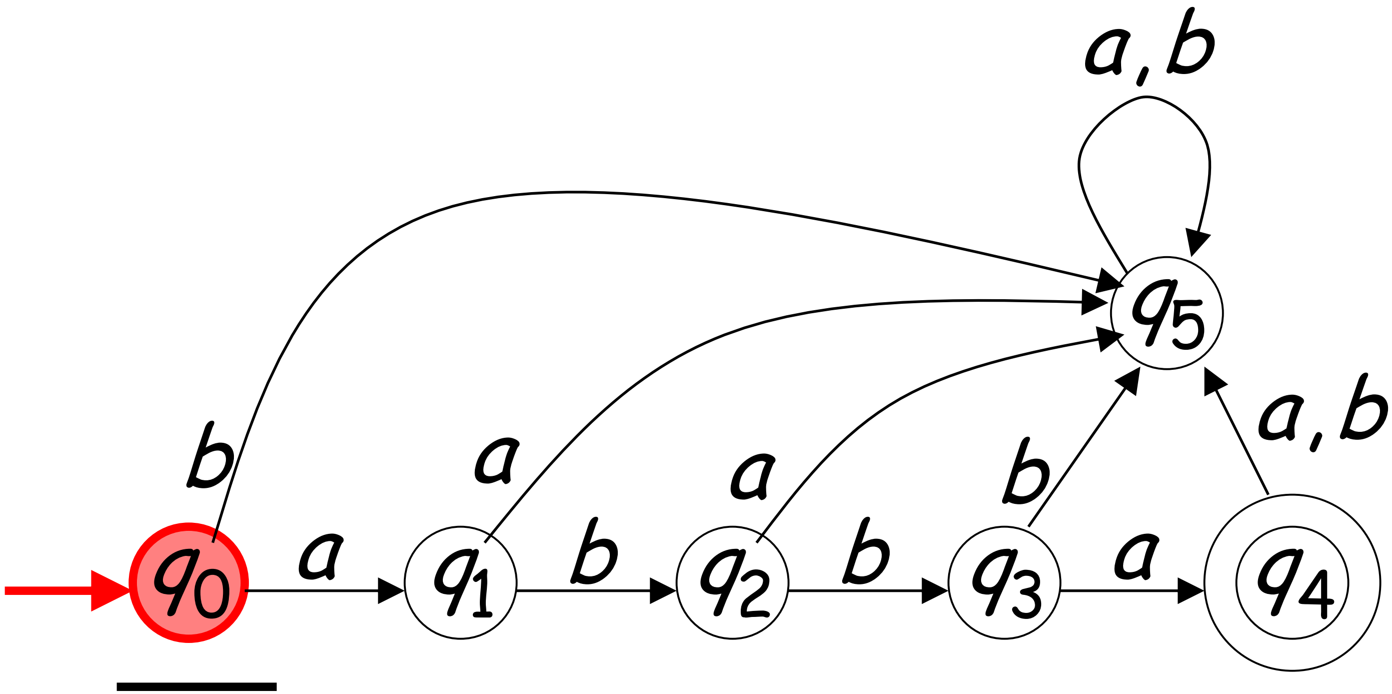
Example

$$\Sigma = \{a, b\}$$



Initial State q_0

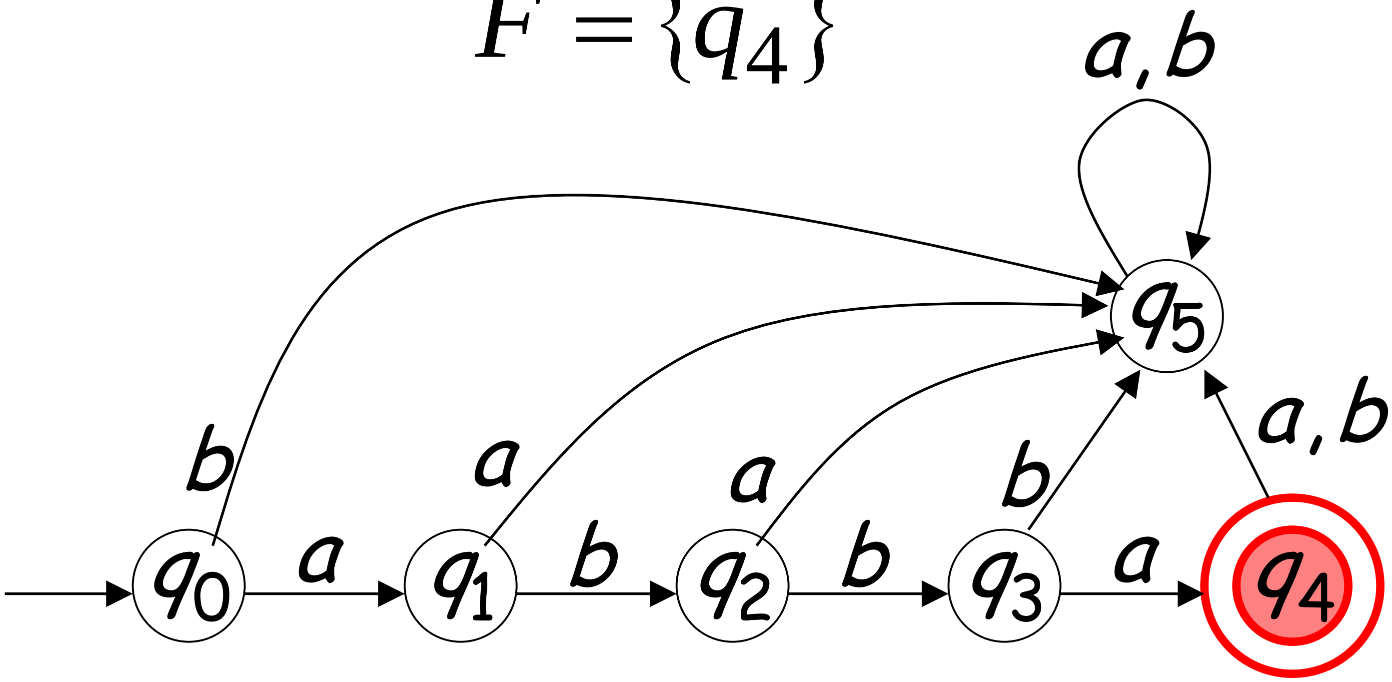
Example



Set of Accepting States $F \subseteq Q$

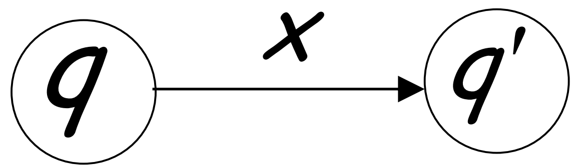
Example

$$F = \{q_4\}$$



Transition Function $\delta : Q \times \Sigma \to Q$

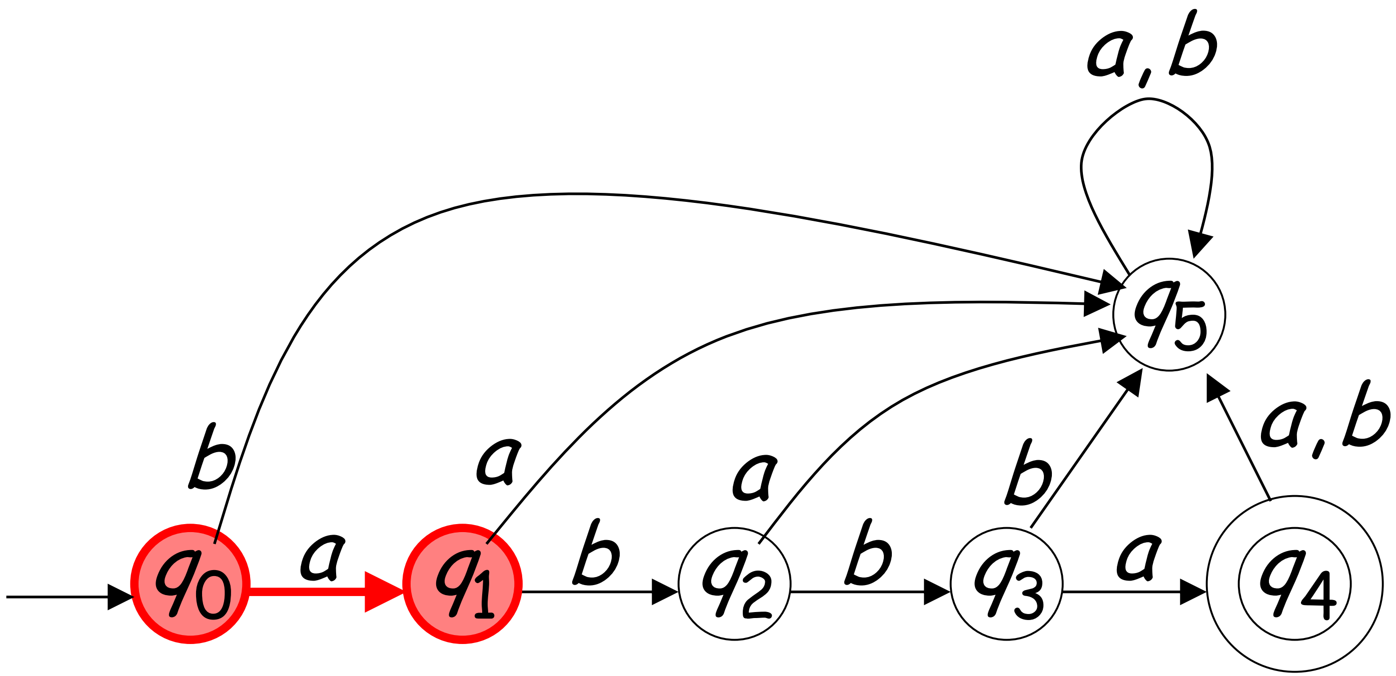
$$\delta(q, x) = q'$$



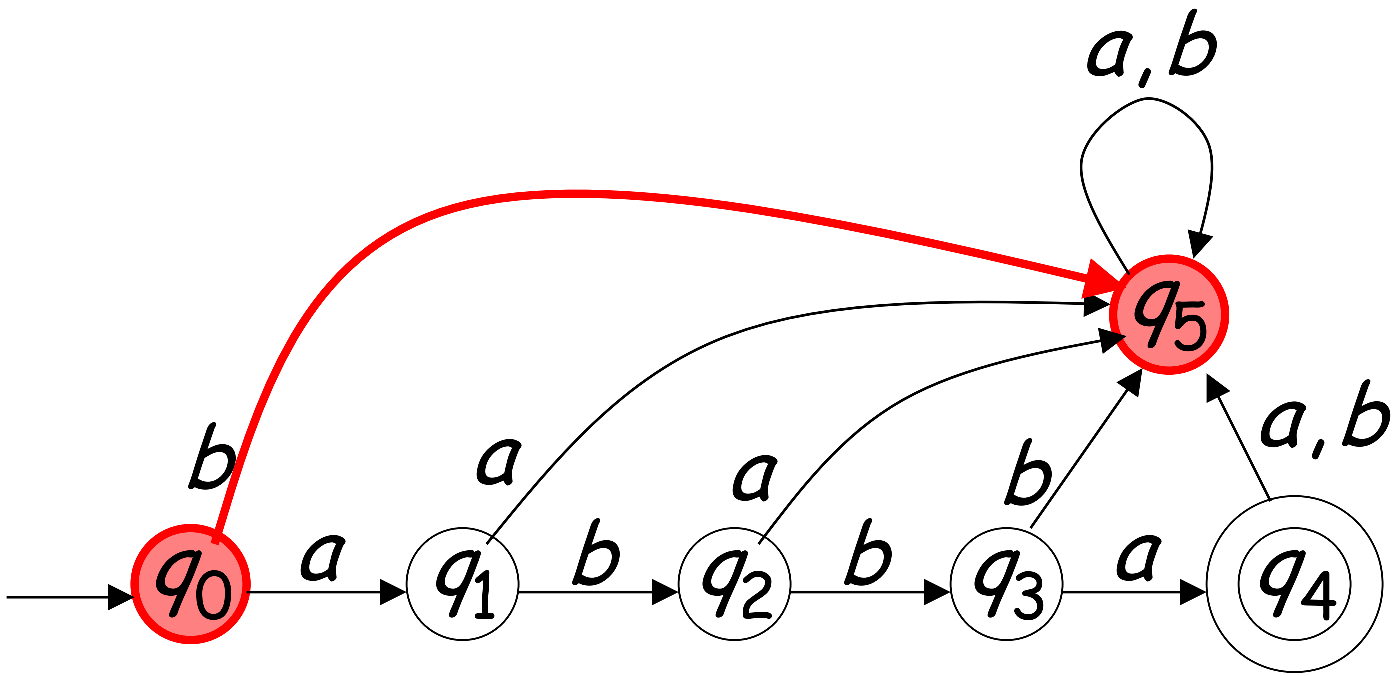
Describes the result of a transition from state q with symbol x

Example:

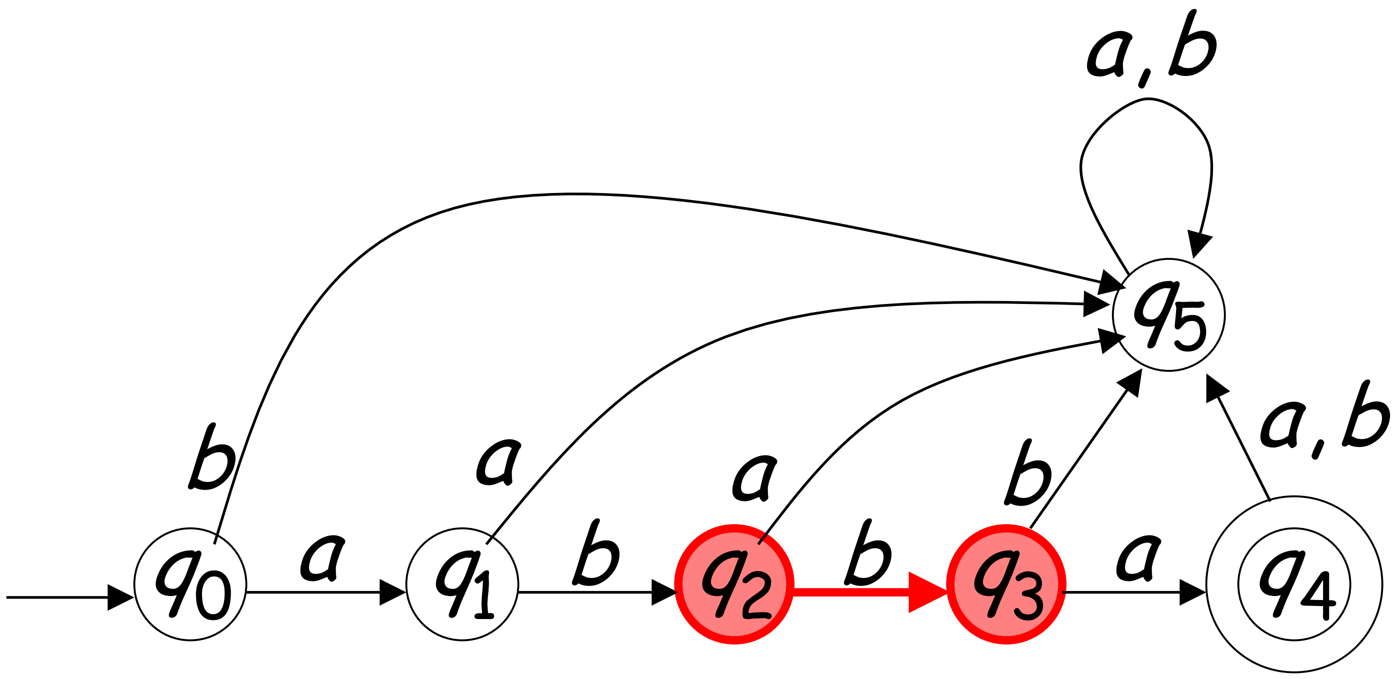
$$\delta(q_0, a) = q_1$$



$$\delta(q_0, b) = q_5$$



$$\delta(q_2, b) = q_3$$

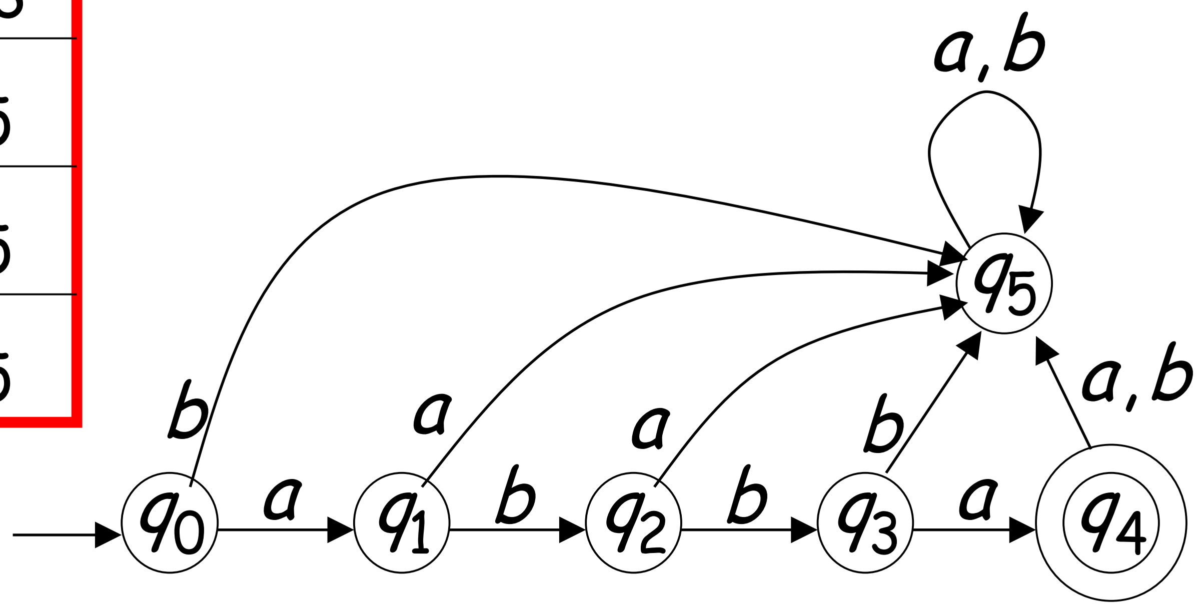


Transition Table for δ

symbols

states

δ	a	b
q_0	q_1	q_5
q_1	q_5	q_2
q_2	q_5	q_3
q_3	q_4	q_5
q_4	q_5	q_5
q_5	q_5	q_5



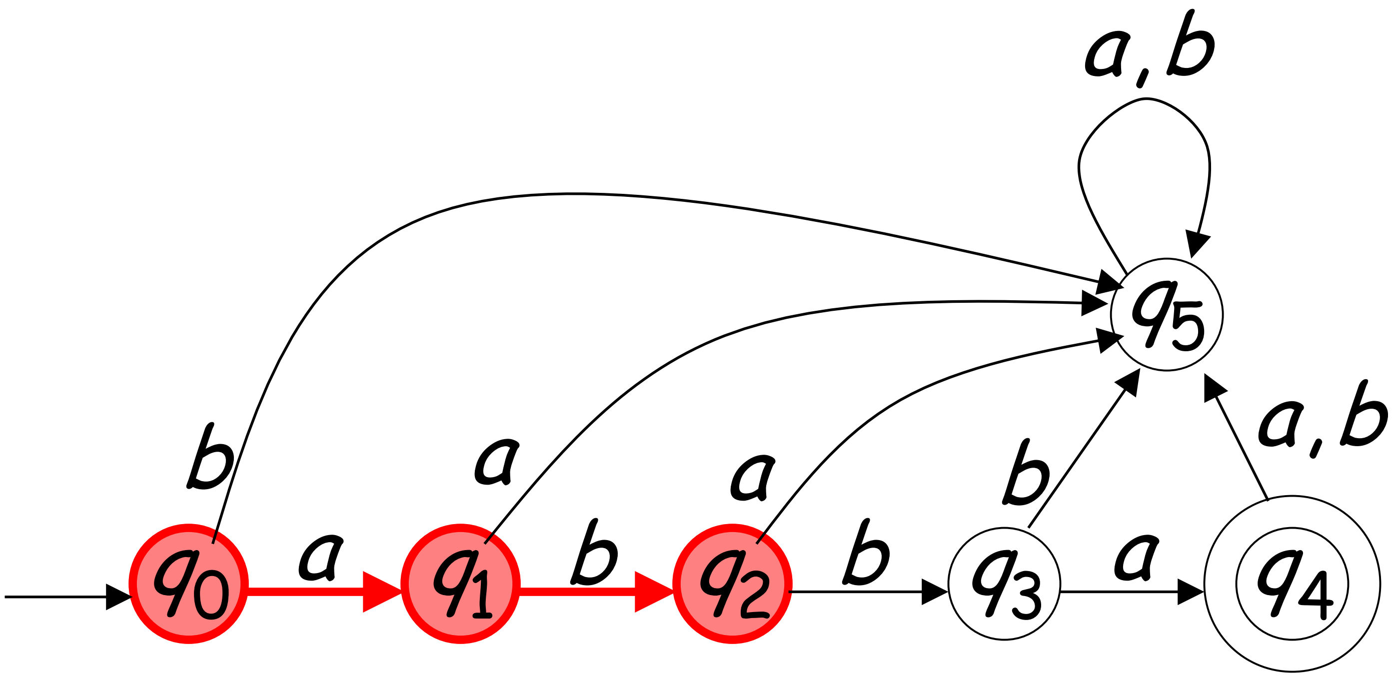
Extended Transition Function

$$\delta^* : Q \times \Sigma^* \to Q$$

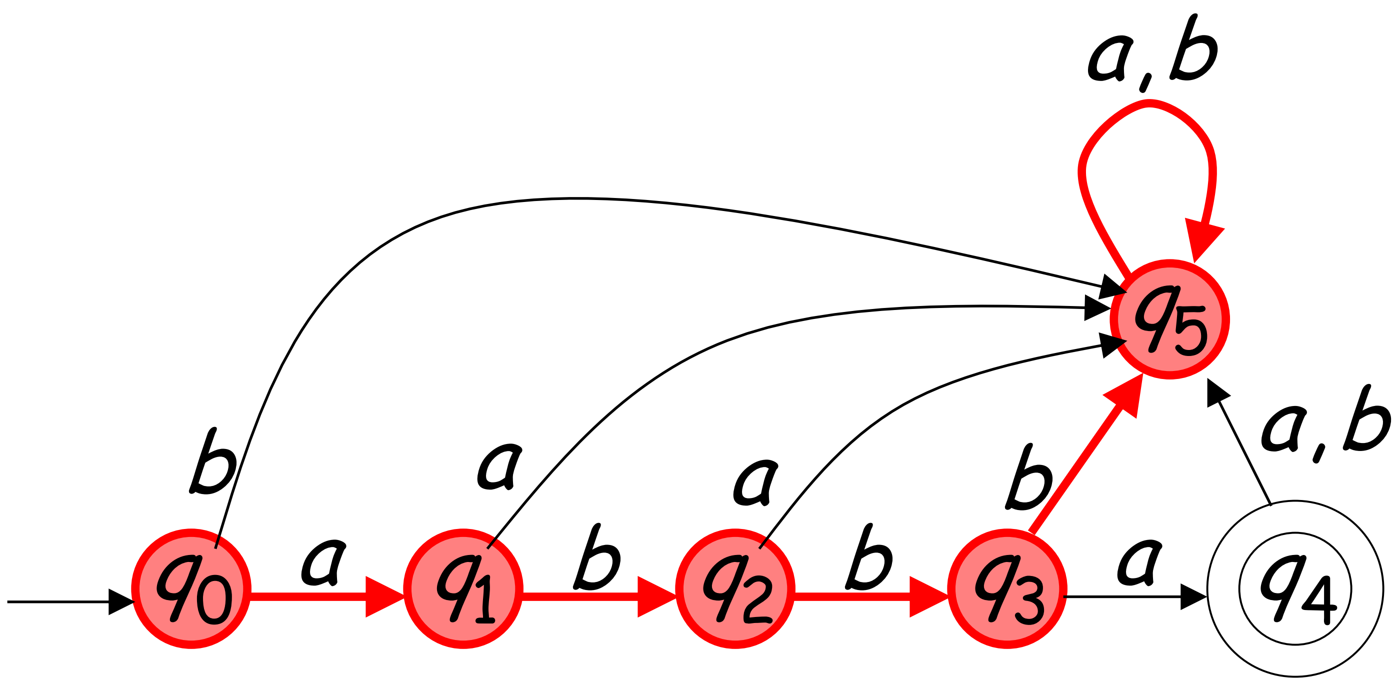
$$\delta^*(q, w) = q'$$

Describes the resulting state after scanning string w from state q

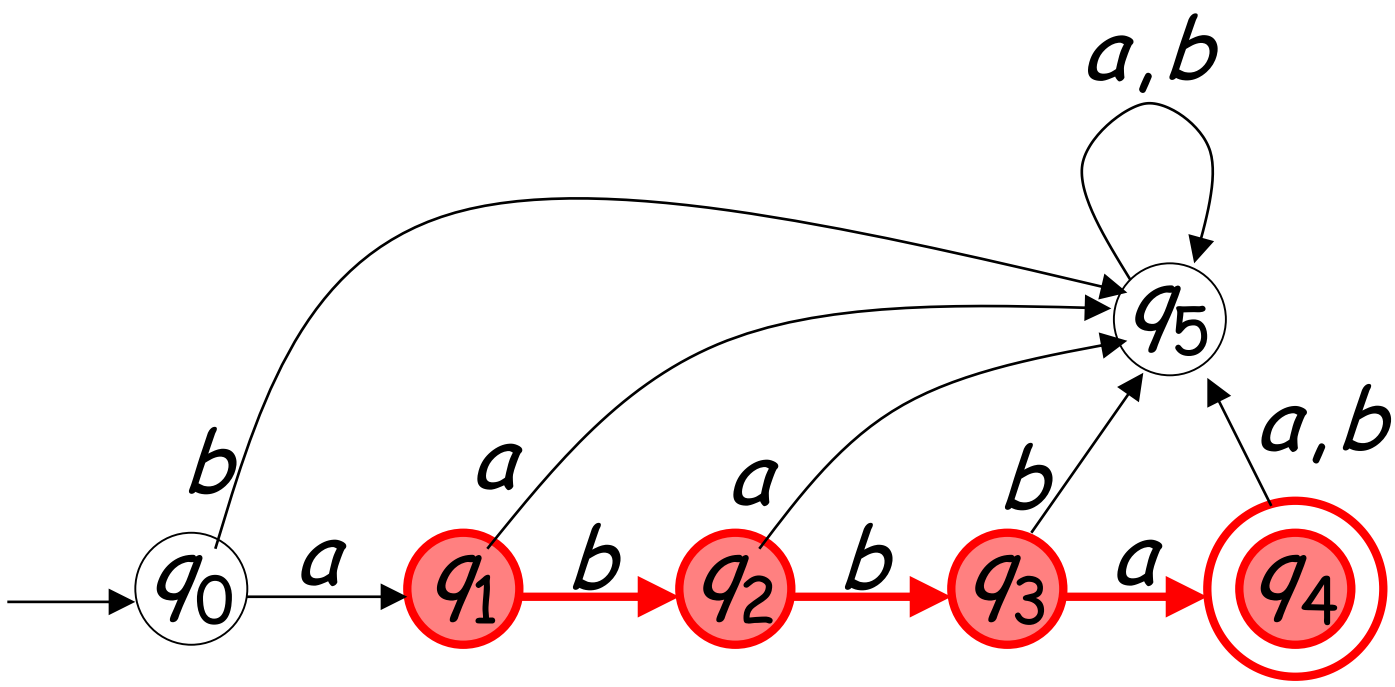
Example: $\delta^*(q_0, ab) = q_2$



$$\delta^*(q_0, abbbaa) = q_5$$



$$\delta^*(q_1, bba) = q_4$$



Special case:

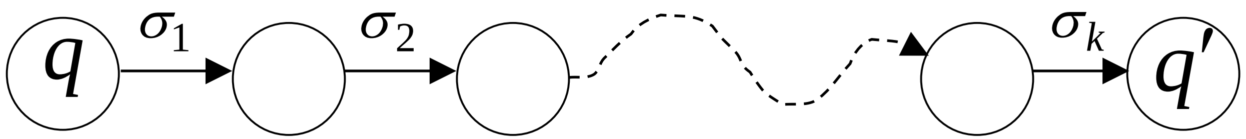
for any state q

$$\delta^*(q,\lambda) = q$$

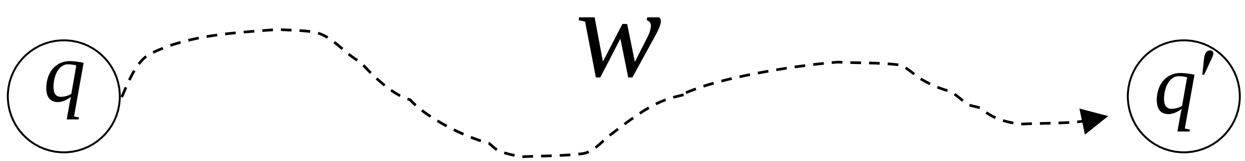
In general: $\delta^*(q,w) = q'$

implies that there is a walk of transitions

$$w = \sigma_1 \sigma_2 \cdots \sigma_k$$



states may be repeated



Language Accepted by DFA

Language of DFA M :

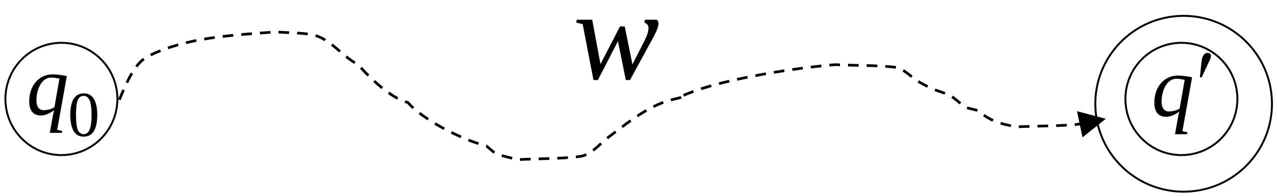
it is denoted as $L(M)$ and contains all the strings accepted by M

We say that a language L' is accepted (or recognized) by DFA M if $L(M) = L'$

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

Language accepted by M:

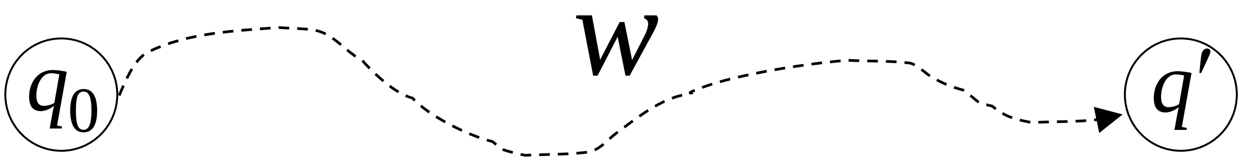
$$L(M) = \{ w \in \Sigma^* : \delta^*(q_0, w) \in F \}$$



$q' \in F$

Language rejected by M:

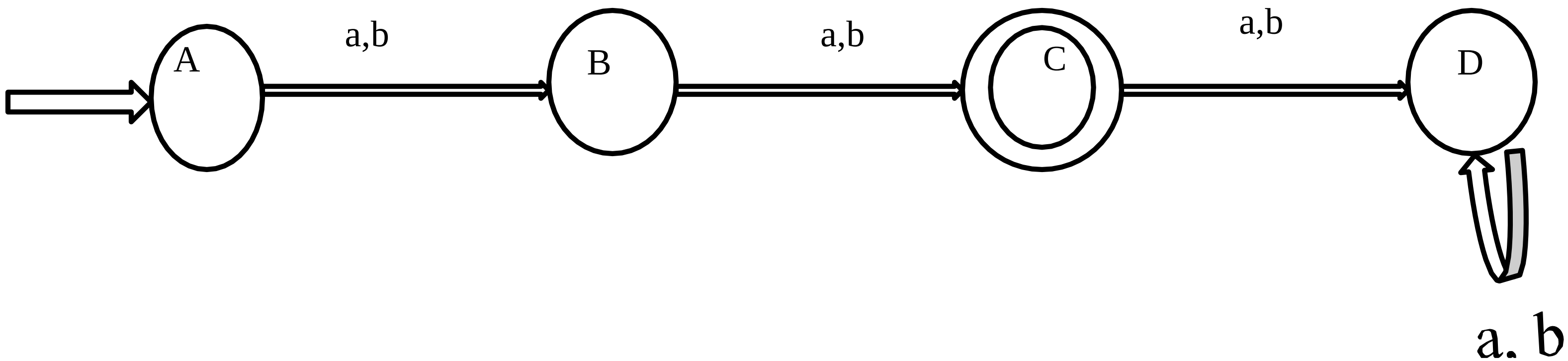
$$\overline{L(M)} = \{ w \in \Sigma^* : \delta^*(q_0, w) \notin F \}$$



$q' \notin F$

Example

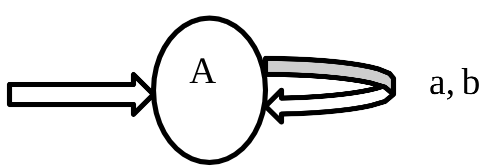
Construct a DFA which accepts set of all string over Σ={a, b} of length 2. L ={aa, ab, ba, bb}



String Accept:-scan the entire string, if we reach a final state from initial state.

language Accept:-All the string in language are "accepted" and all string which are not in the language are "rejected"

Example

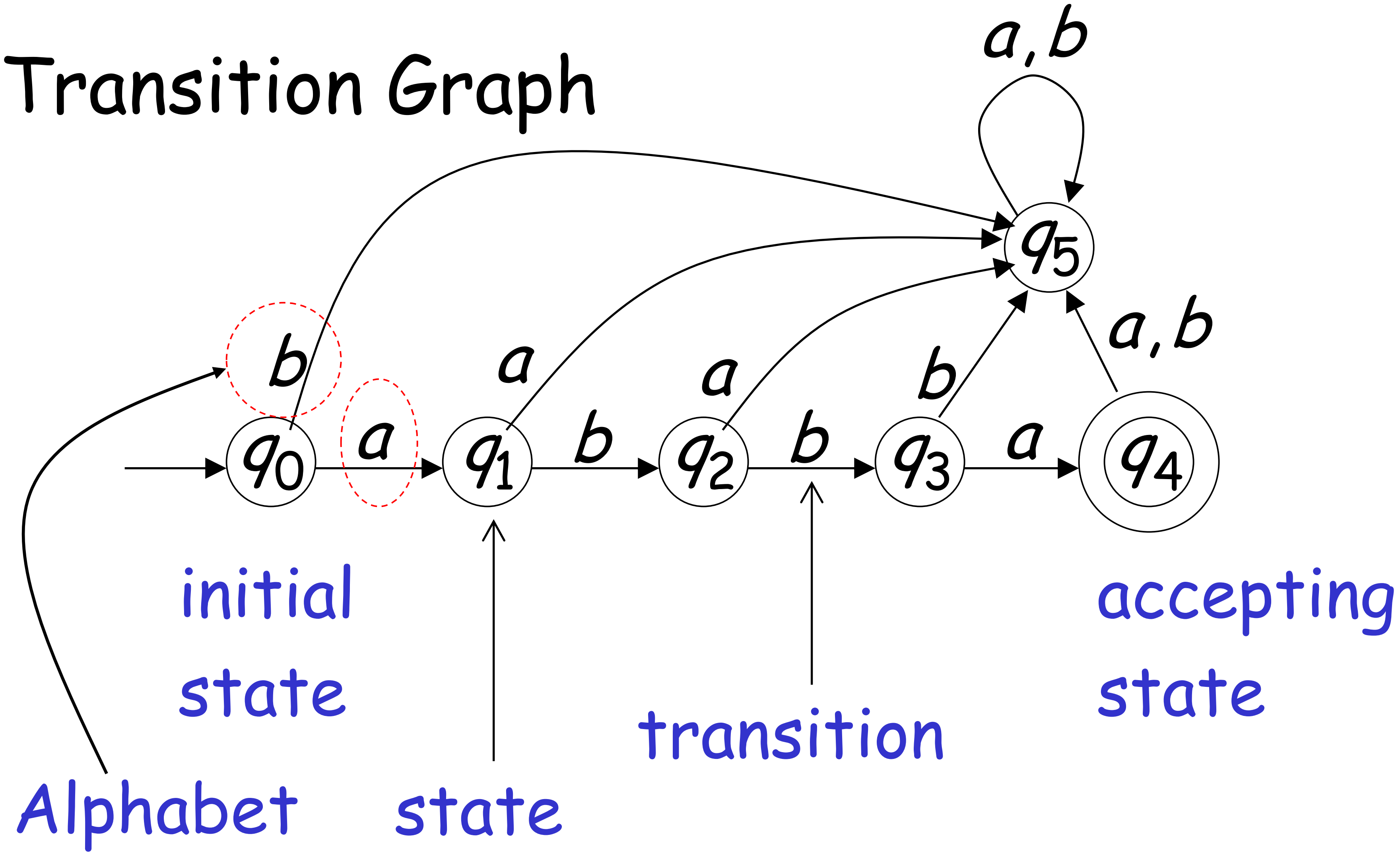


Above DFA is incorrect because it also accept the string which is not in language

Review

- Language
- Language Operations
- * (Kleene closure) and + (positive closure)
- Formal Definition of Finite Automata
- Language Accepted by Finite Automata

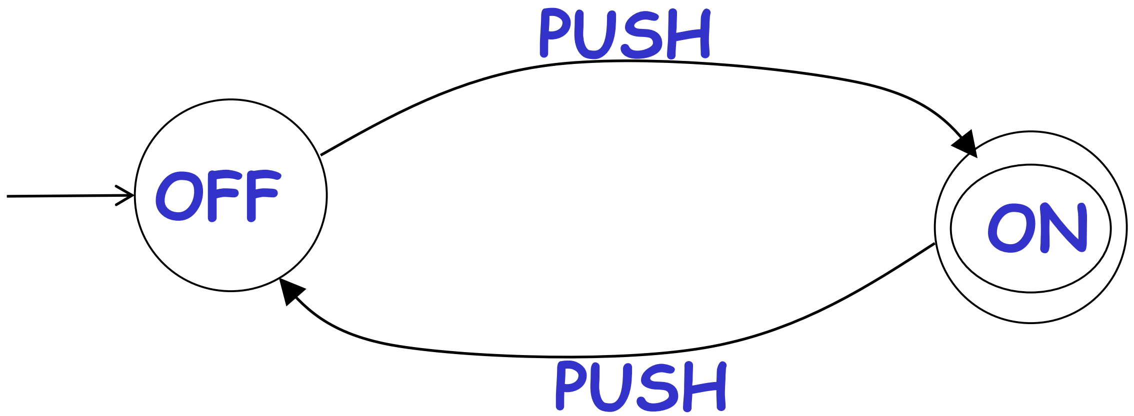
Transition Graph



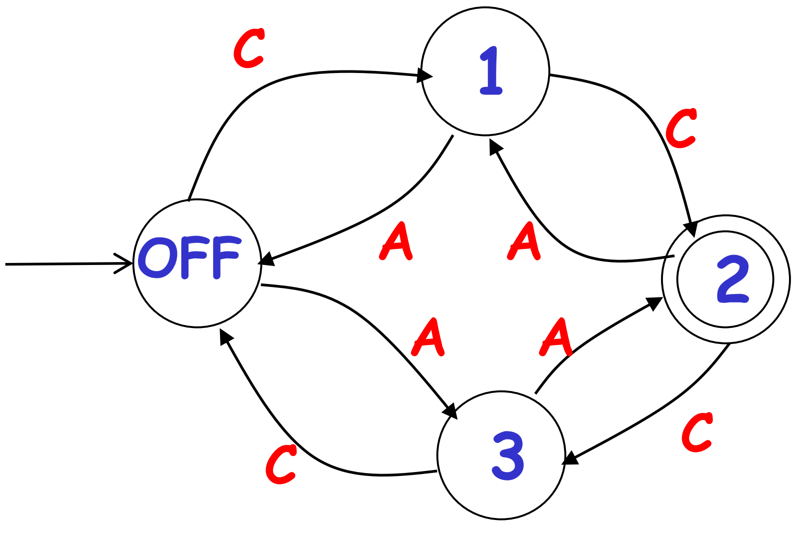
$\Sigma = \{a, b\}$

For every state, there is a transition for every symbol in the alphabet

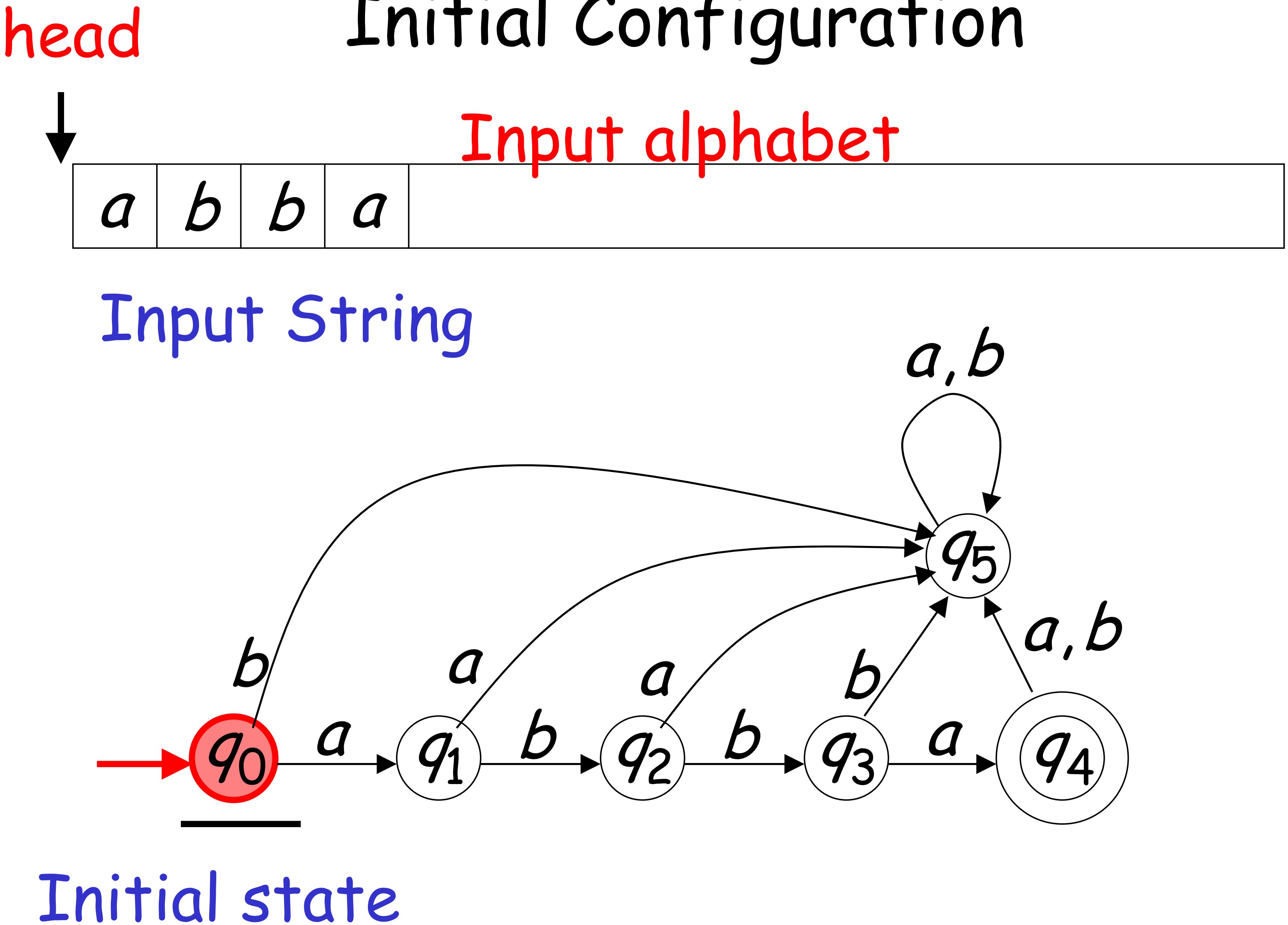
•Electric Switch



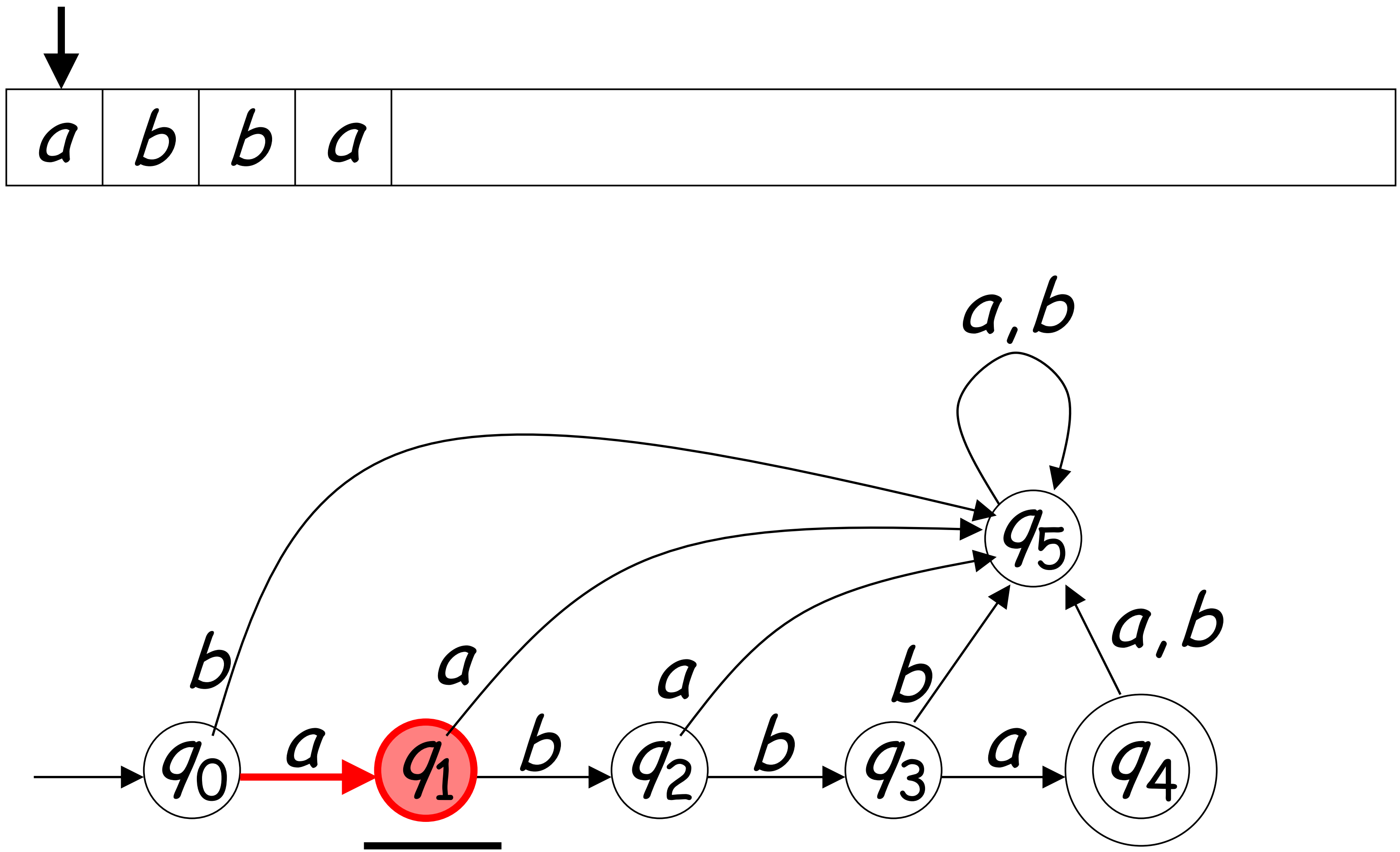
•FAN REGULATOR

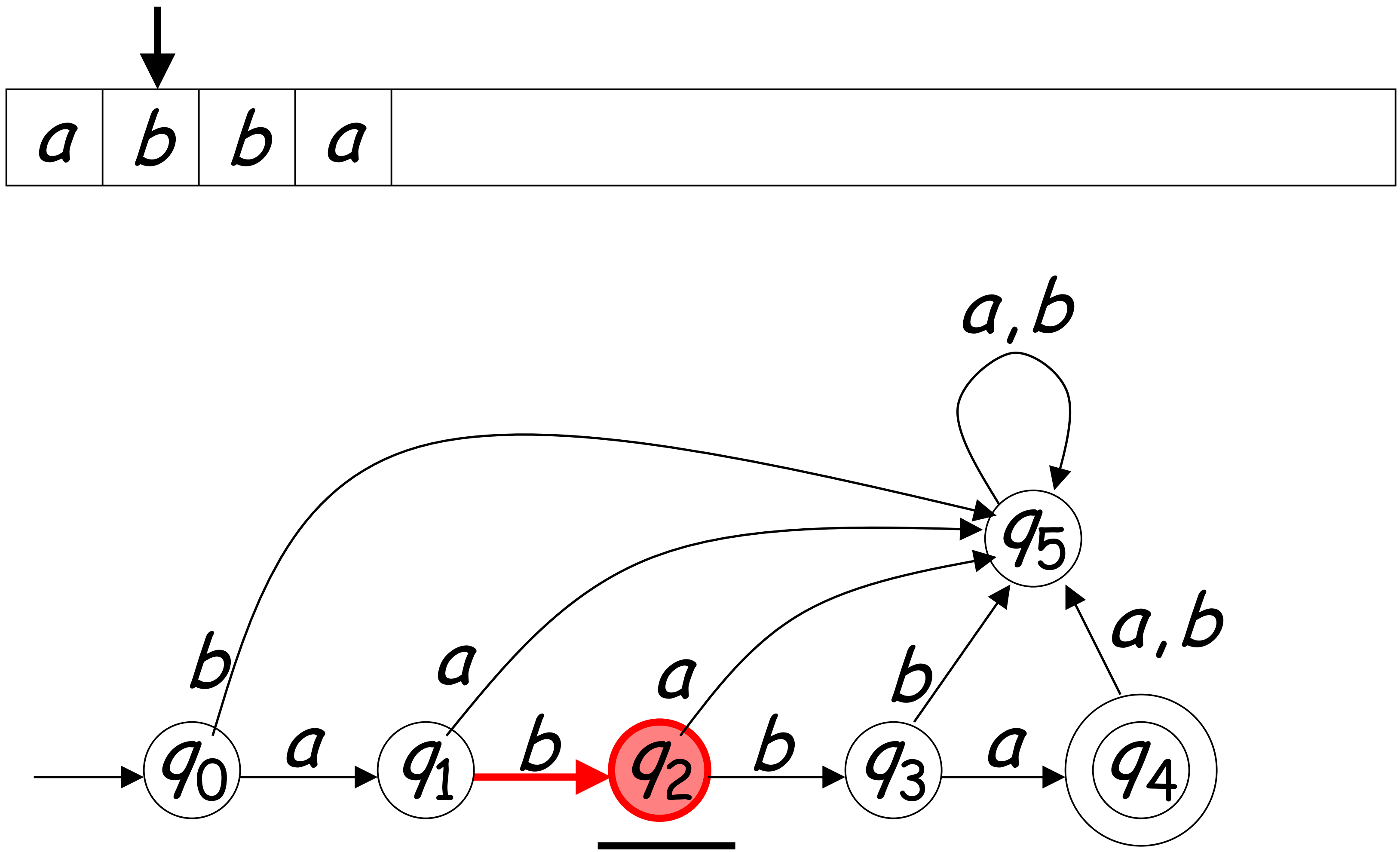


Initial Configuration

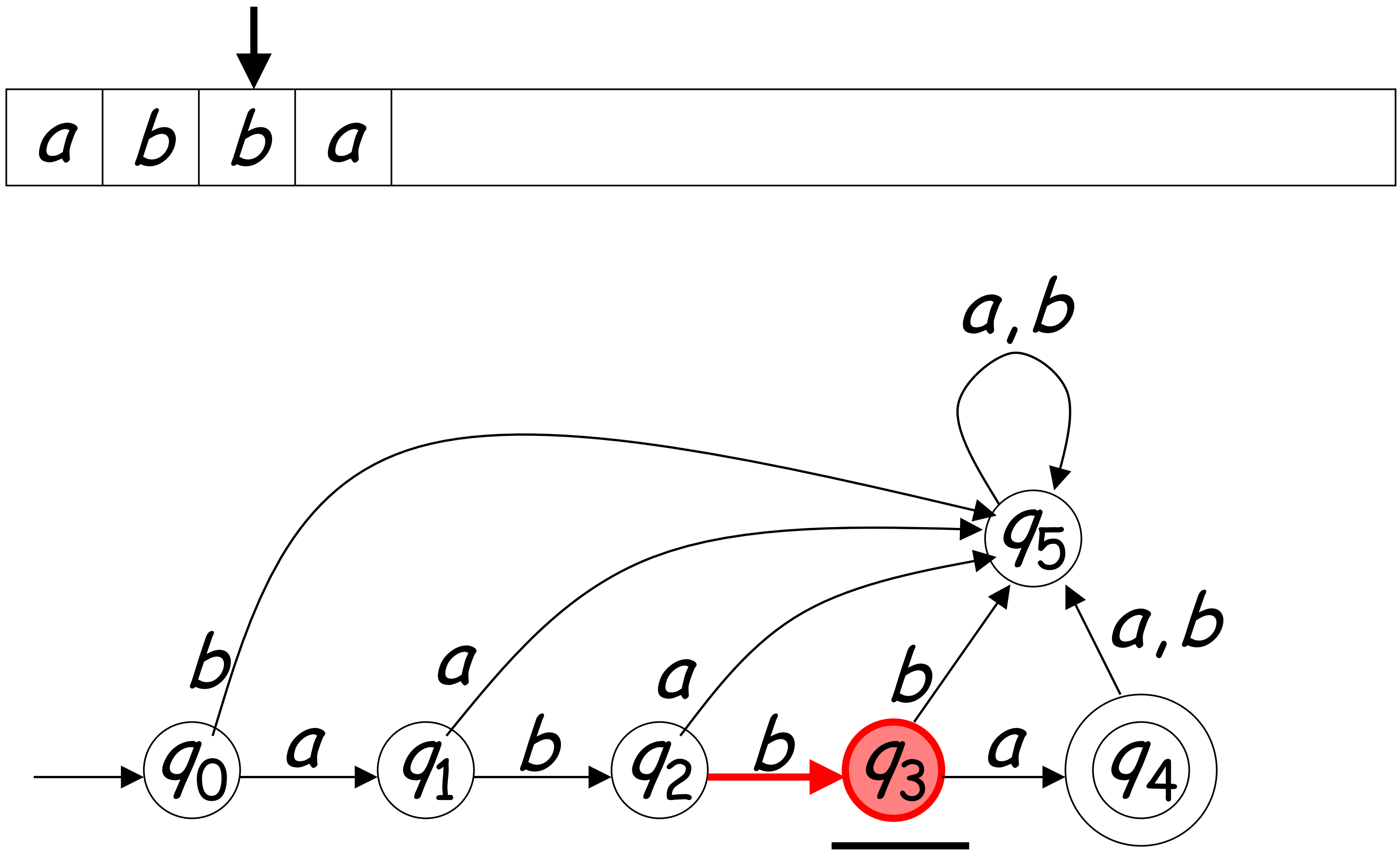


Scanning the Input





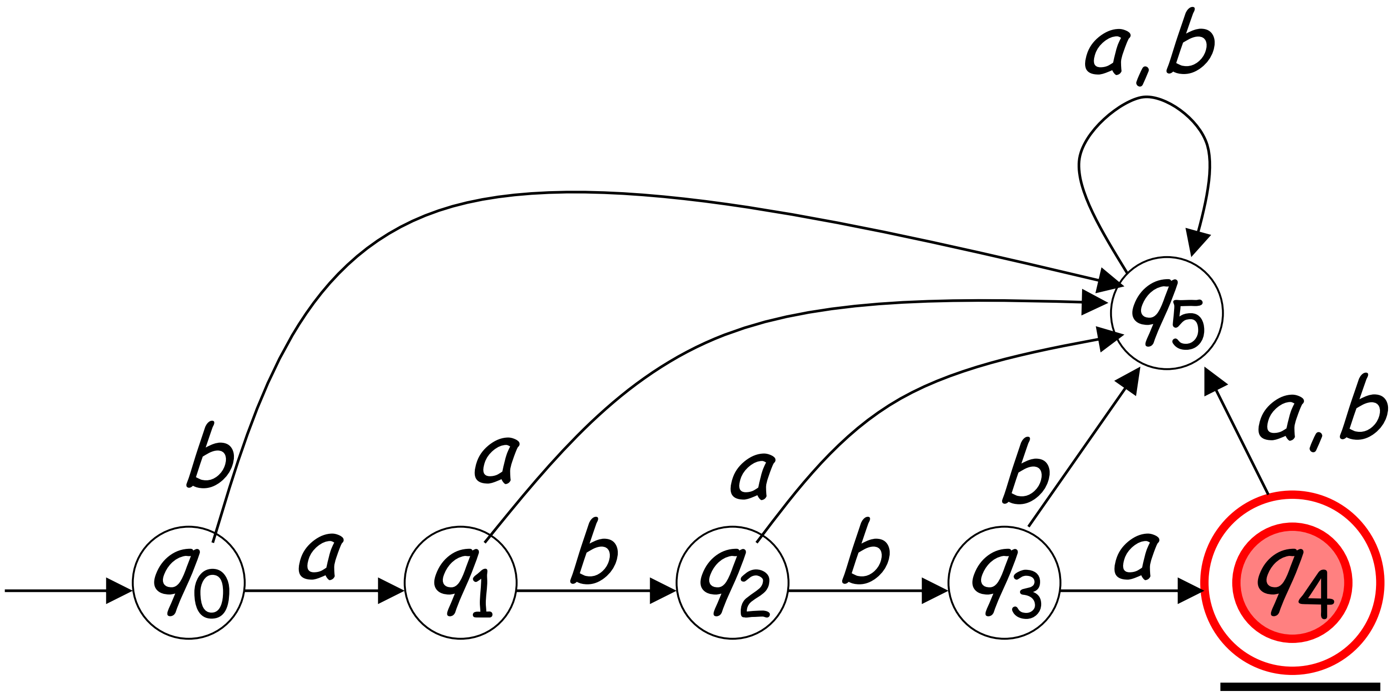
a
b
b
a
a, b
b
a
a
a
b
b
a
a, b
q0
q1
q2
q3
q4
q5



a b b a
a,b
b
a
a
a,b
b
a
b
b
a
q0 q1 q2 q3 q4 q5

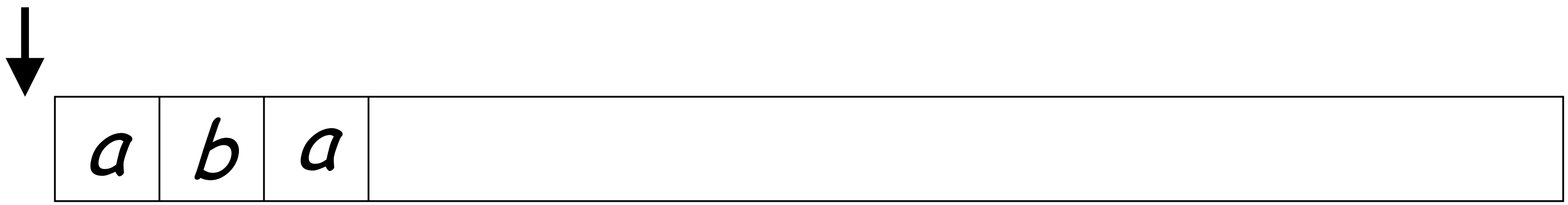
Input finished

a	b	b	a	

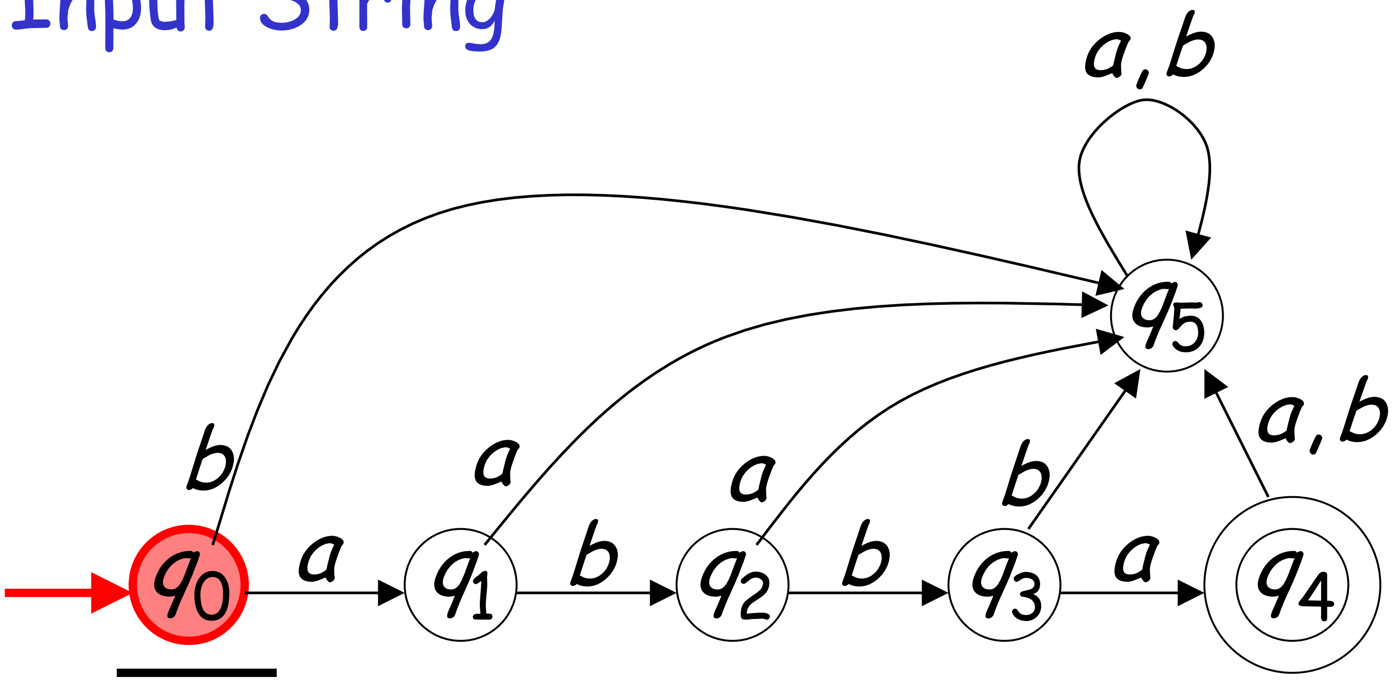


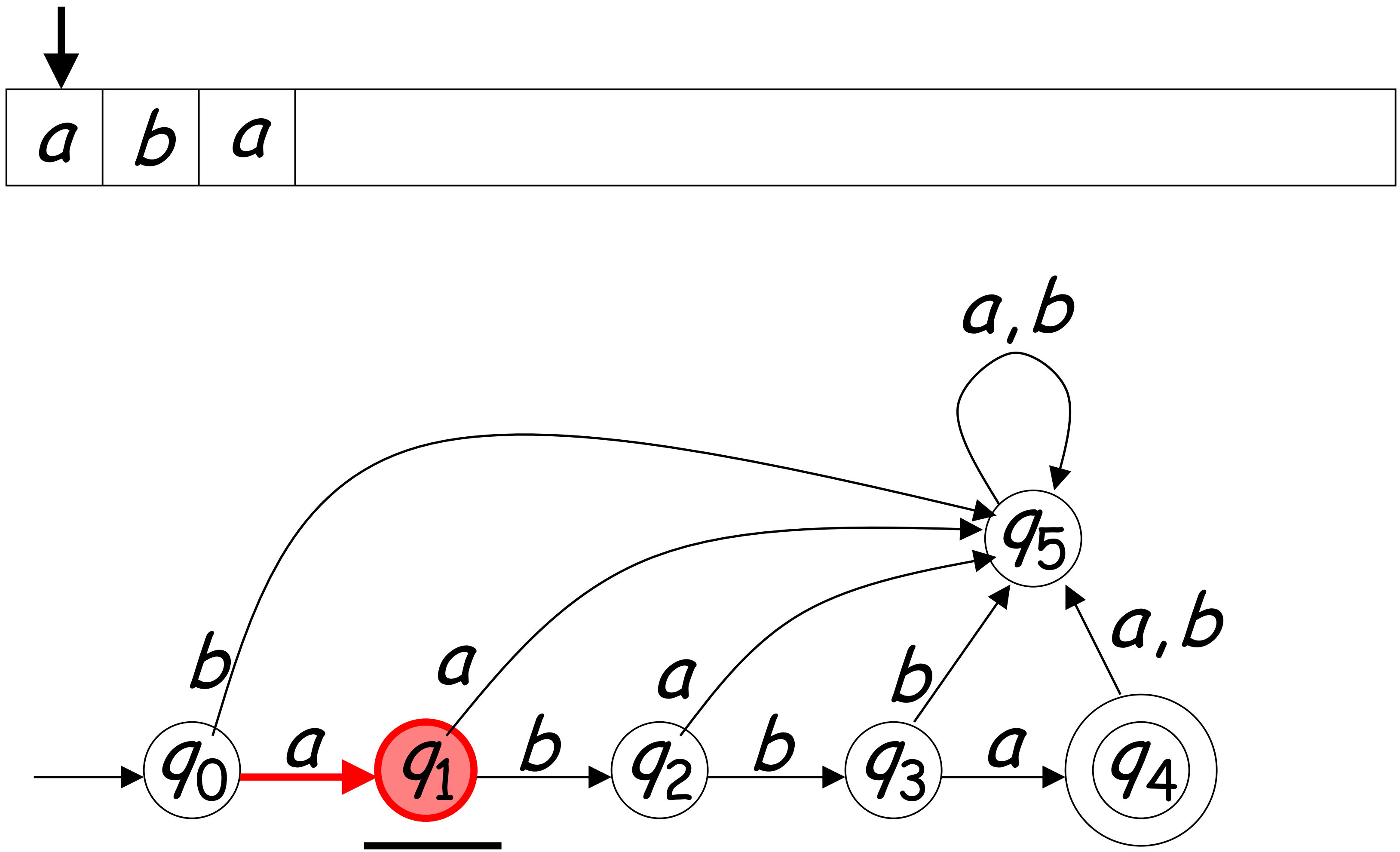
accept

A Rejection Case

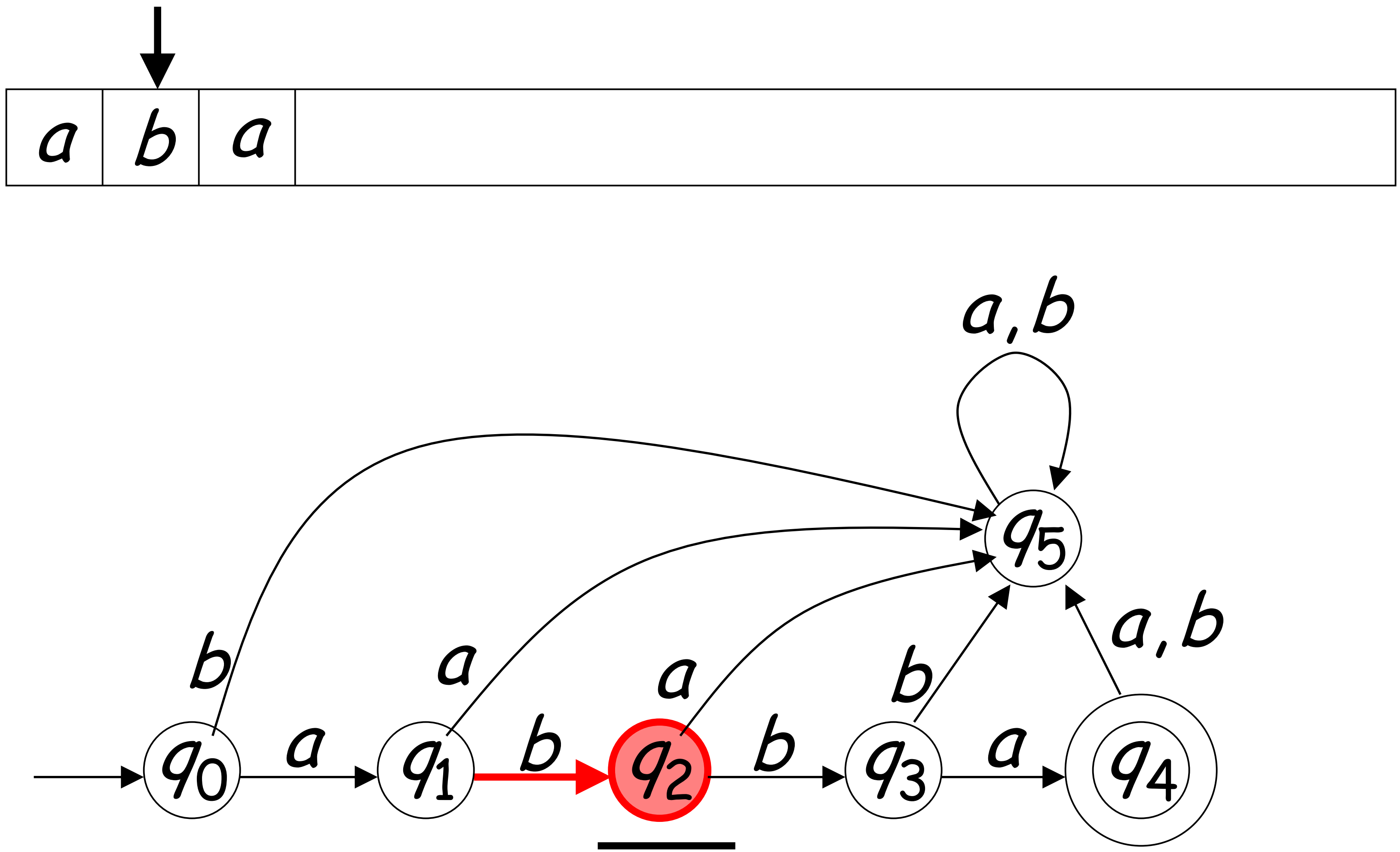


Input String



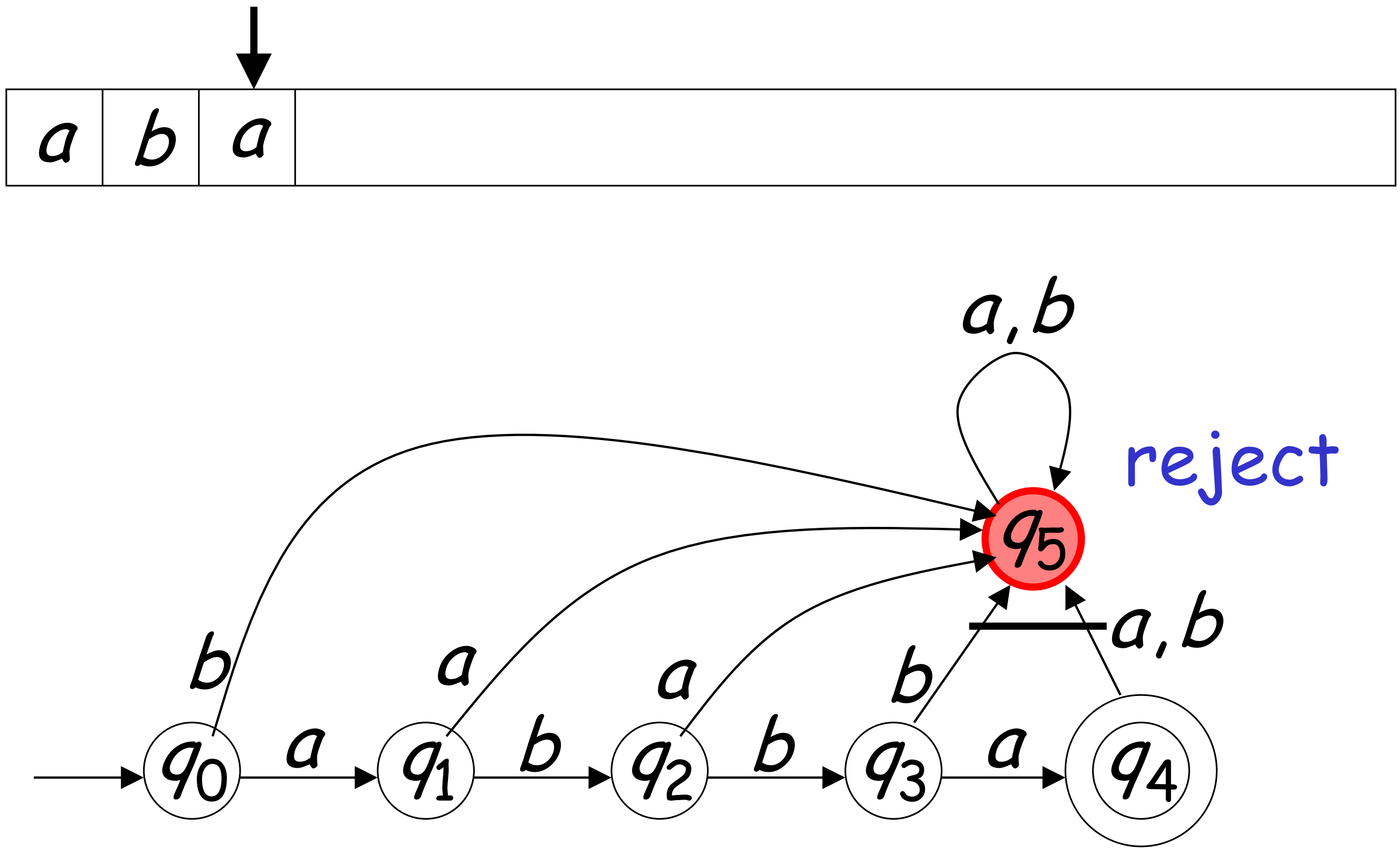


a
b
a
a, b
b
a
a
b
a, b
q_0
a
q_1
b
q_2
b
q_3
a
q_4
q_5



a b a
a,b
q_5
b
a
a
b
a,b
q_0
a
q_1
b
q_2
b
q_3
a
q_4

Input finished

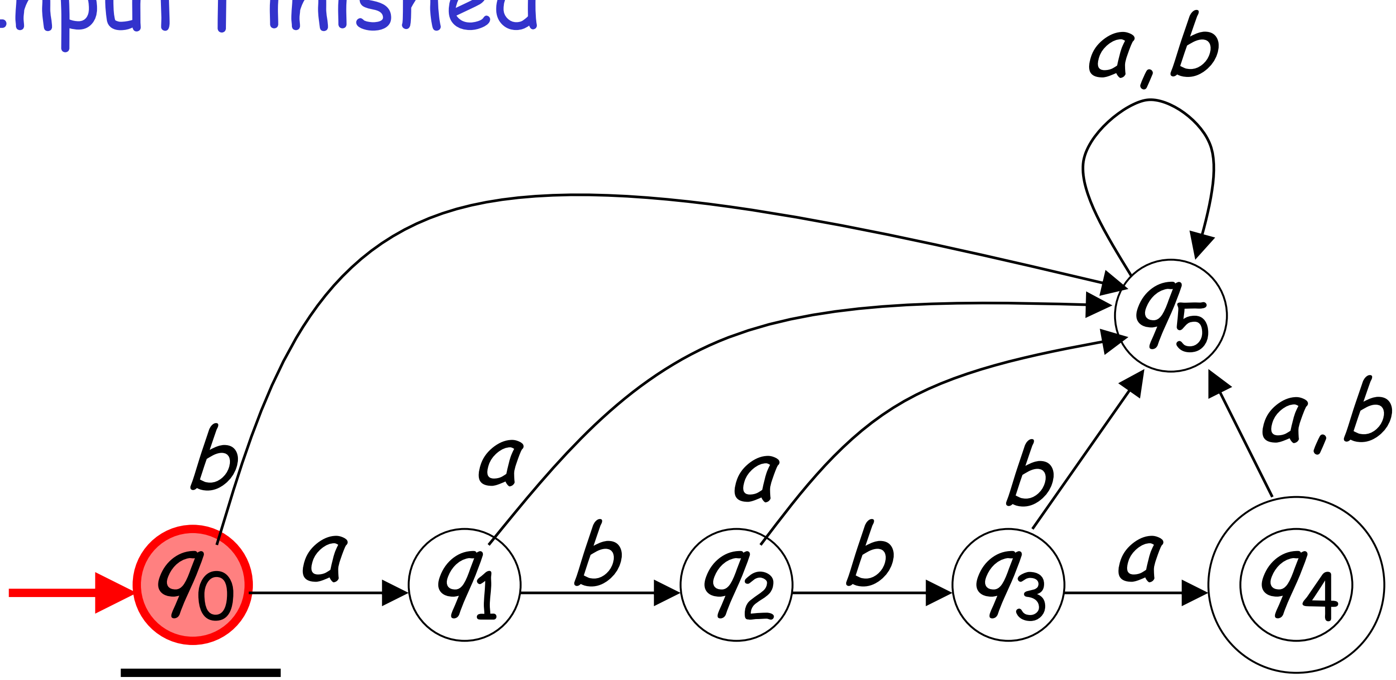


Another Rejection Case

Input String is empty

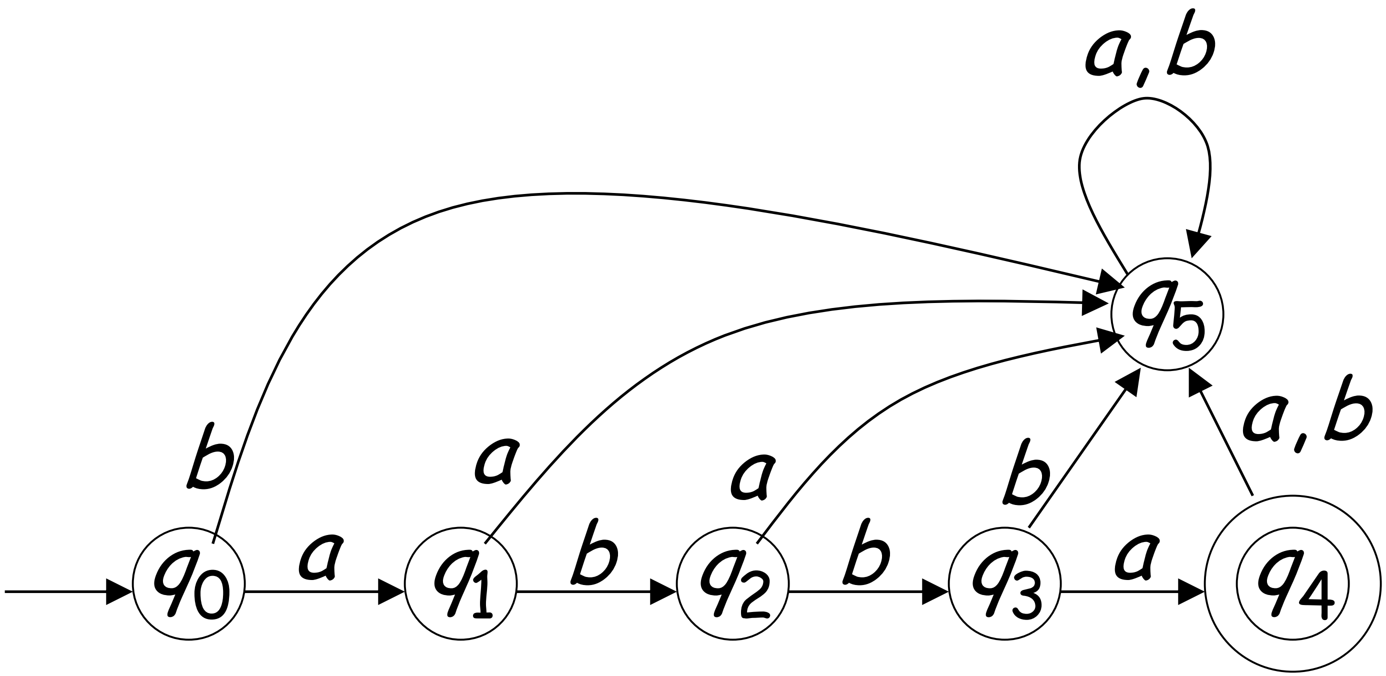
(λ)

Input Finished



reject

Language Accepted: $L = \{abba\}$



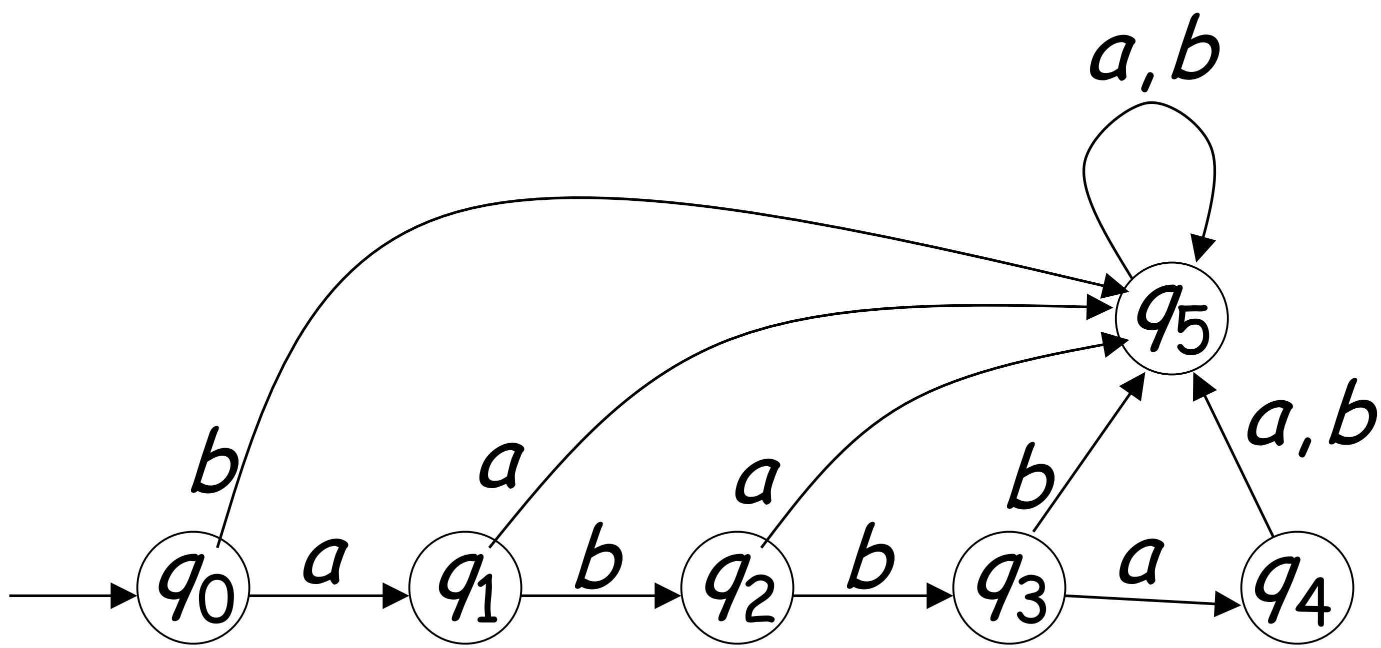
To accept a string:

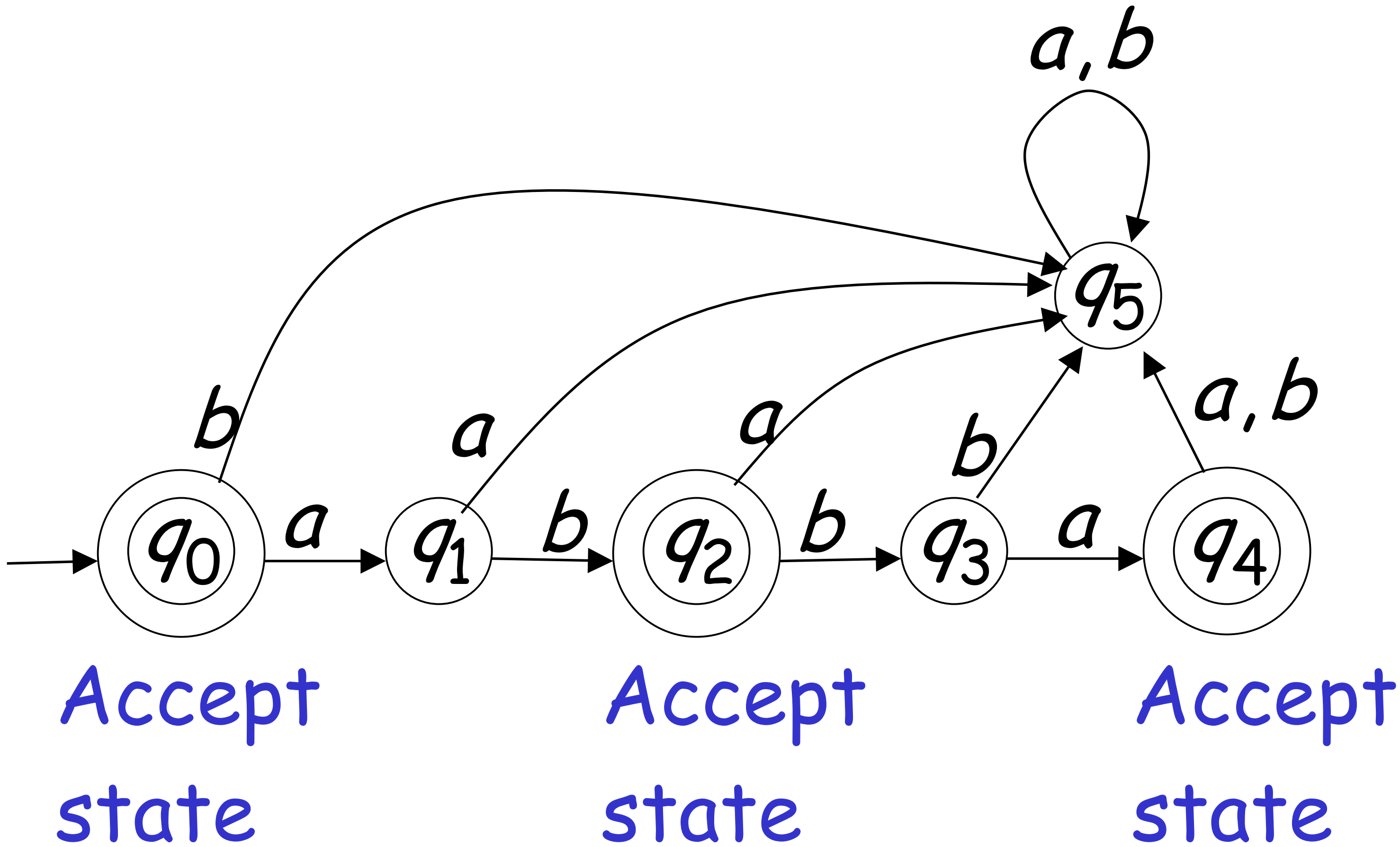
all the input string is scanned
and the last state is accepting

To reject a string:

all the input string is scanned
and the last state is non-accepting

Language Accepted: $L = \{\lambda, ab, abba\}$

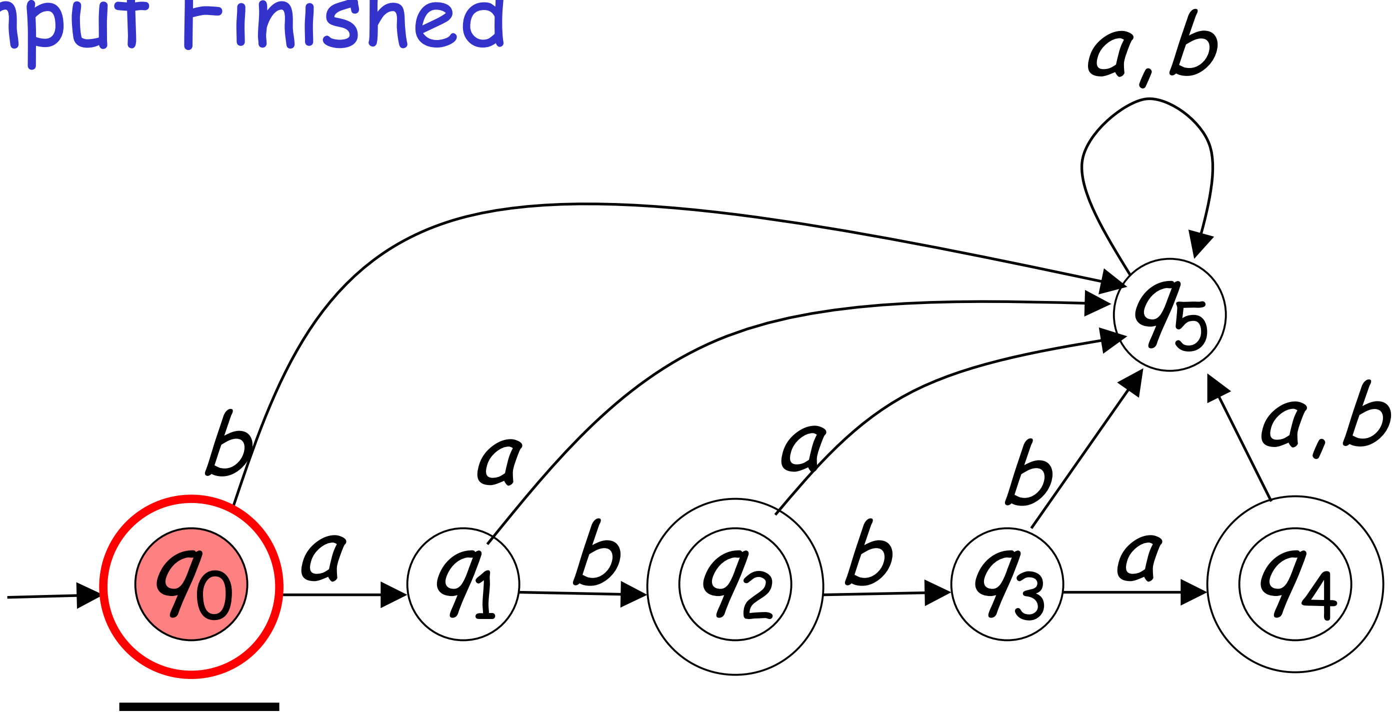




a,b
b
a
a
b
a,b
q0
a
q1
b
q2
b
q3
a
q4
q5
Accept state
Accept state
Accept state

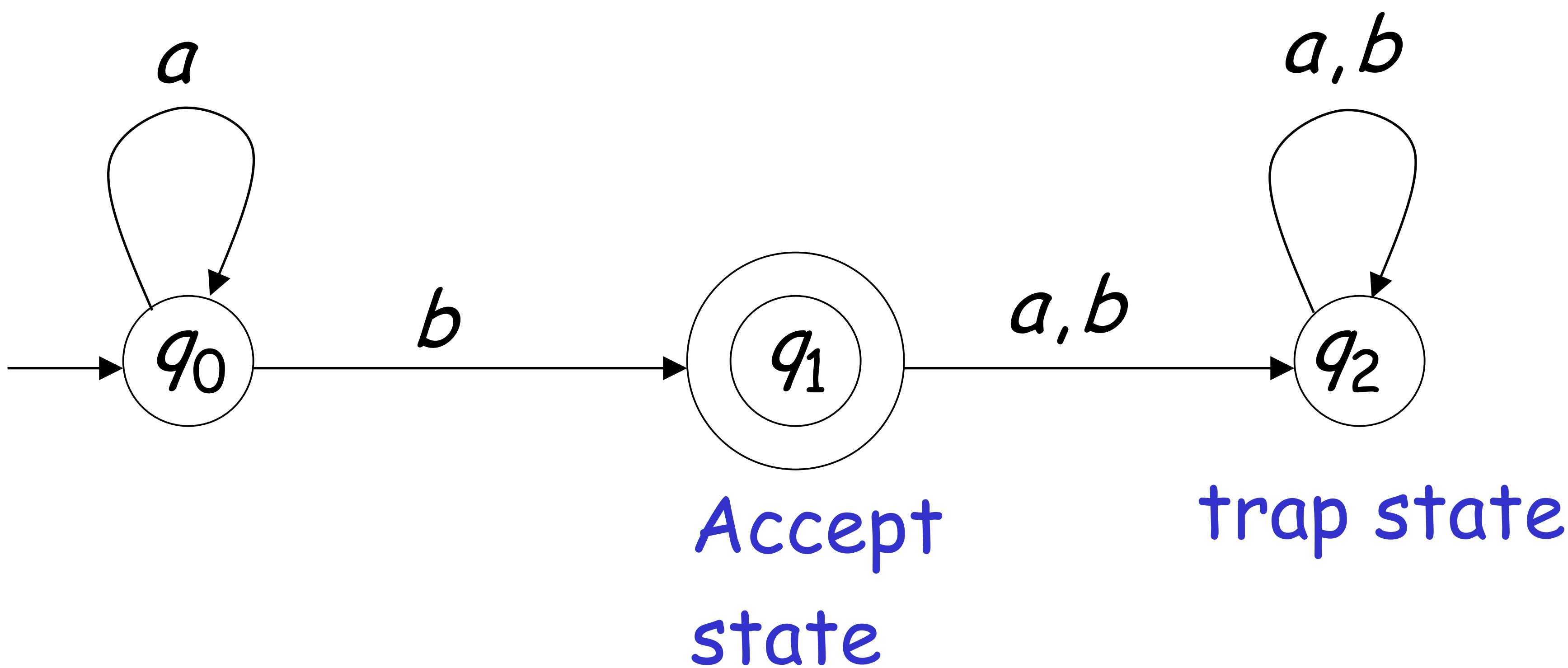
Empty Tape

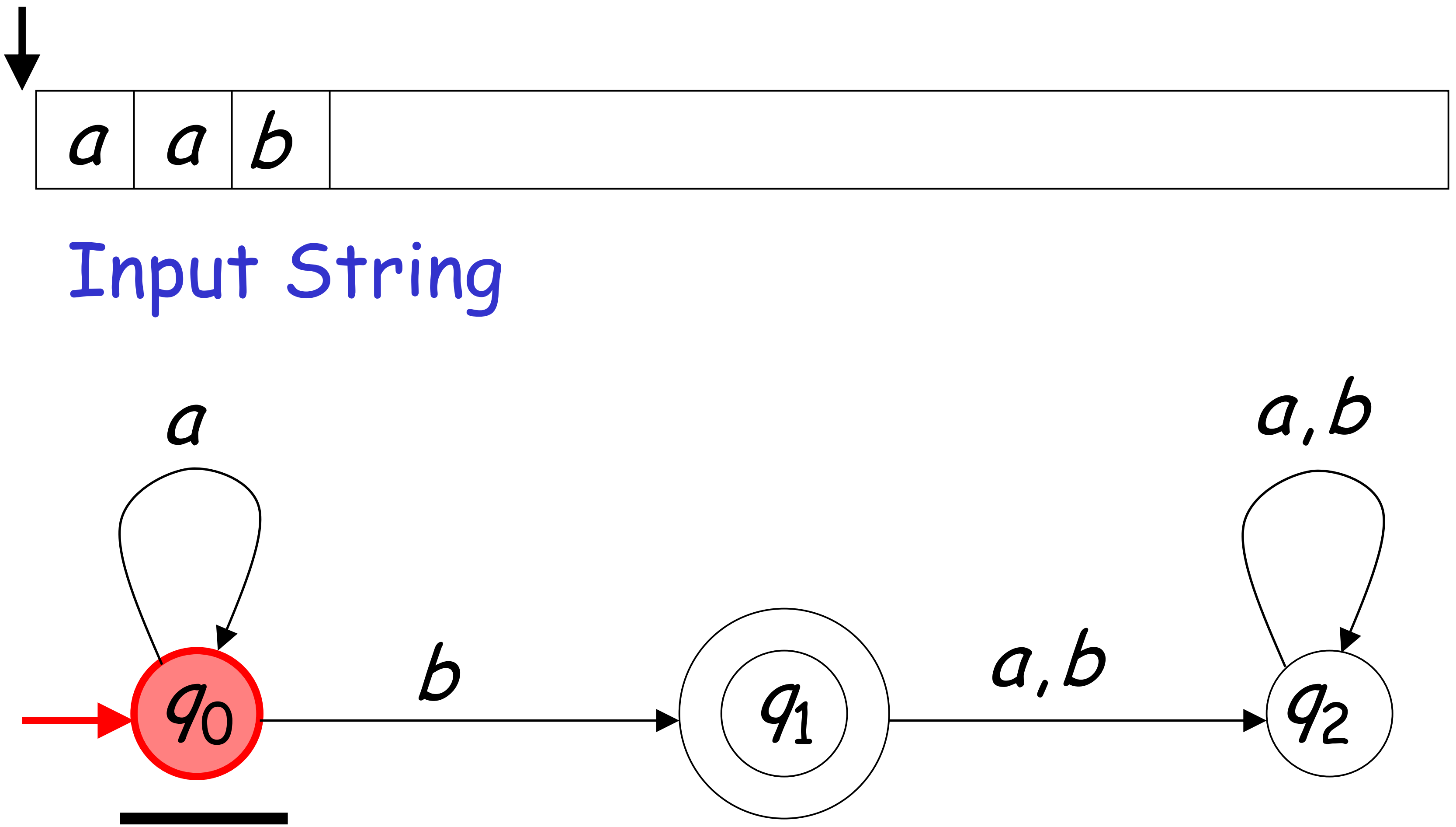
Input Finished



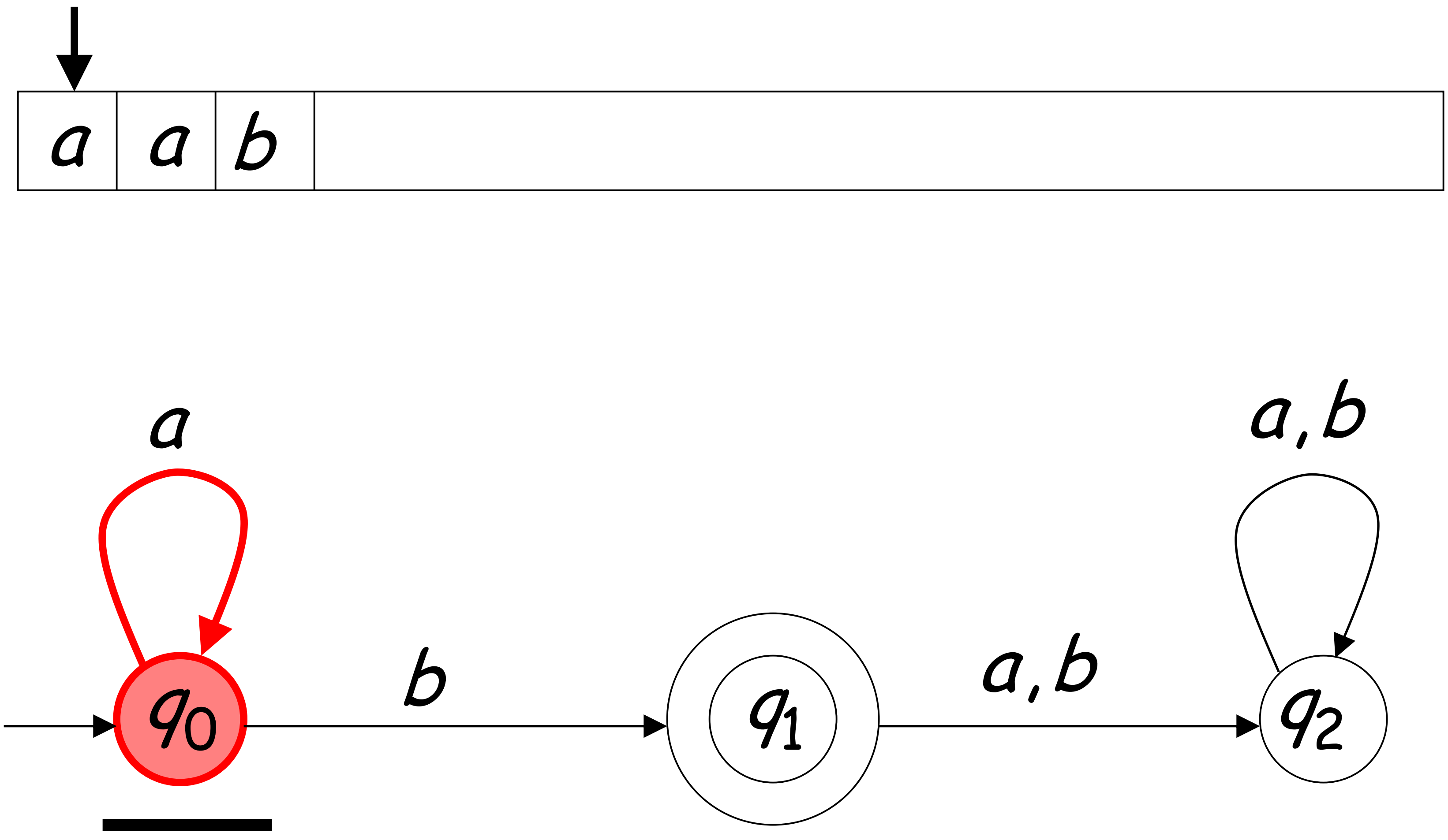
accept

Another Example

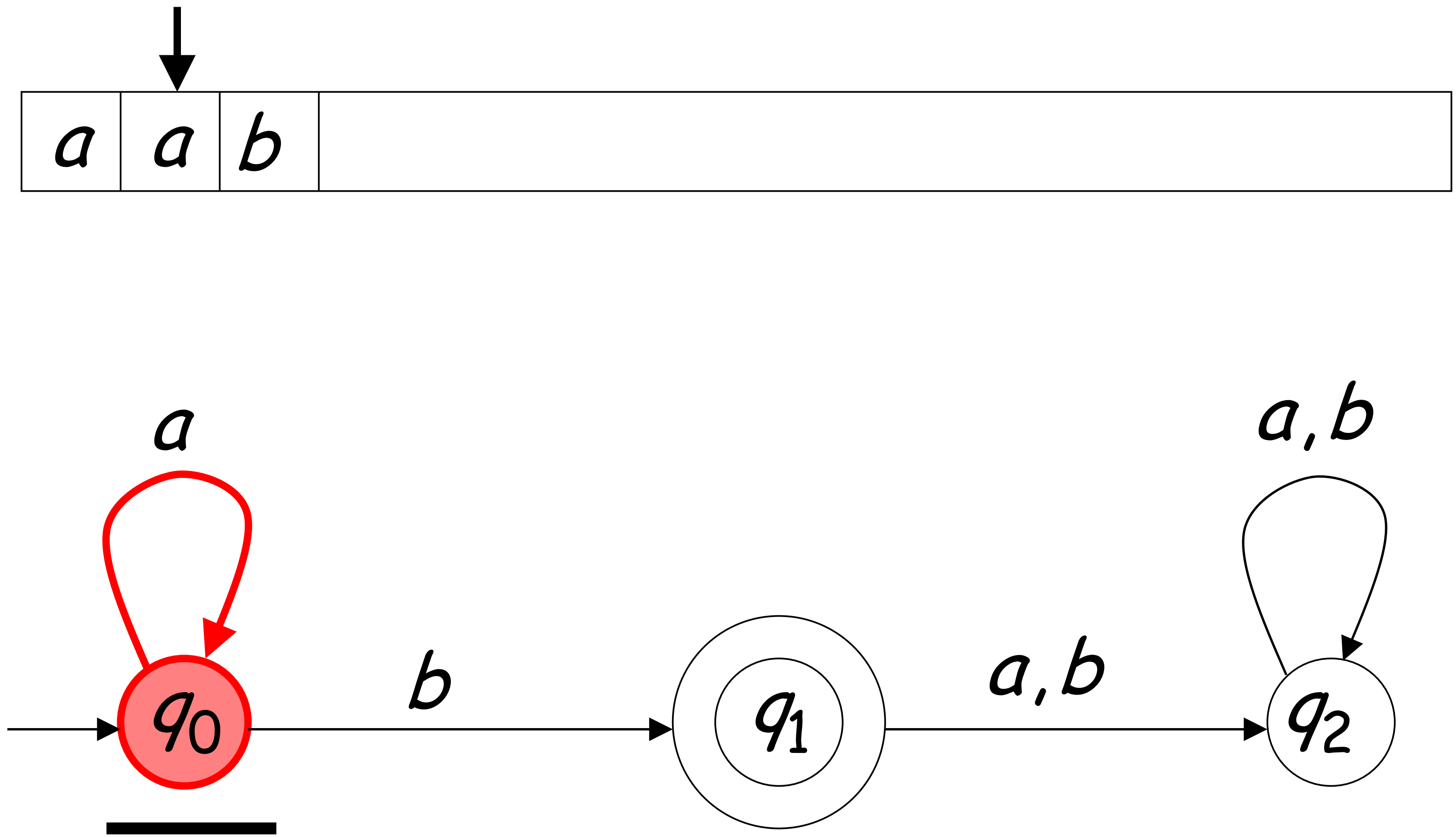




a a b
Input String
a
b
a,b
a,b
q0
q1
q2

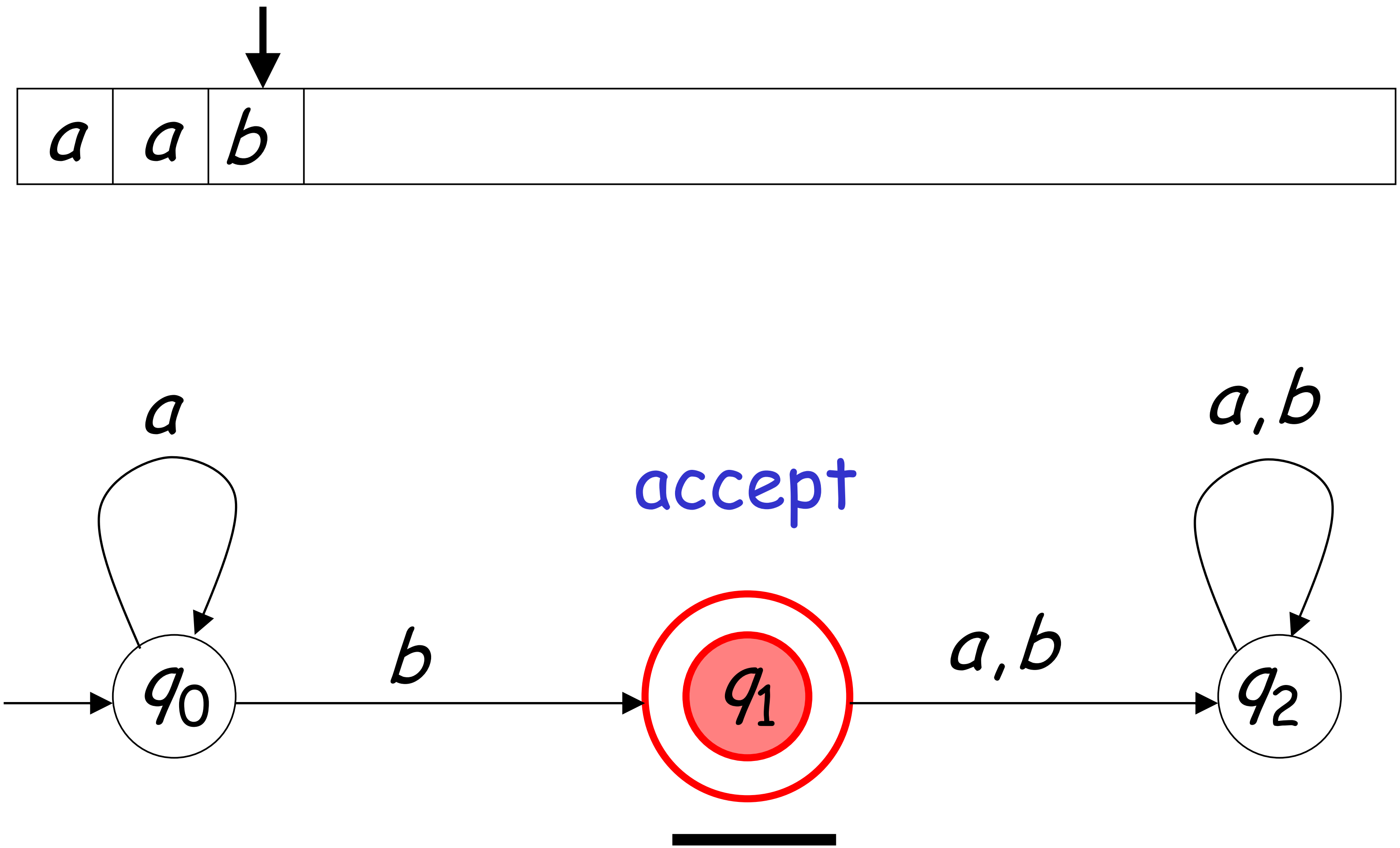


a
a
b
a
b
a,b
a,b
q_0
q_1
q_2

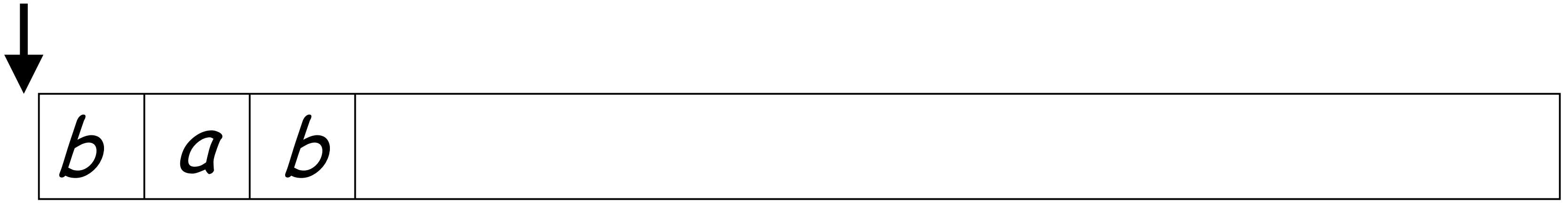


a
a
b
a
a,b
b
a,b
q_0
q_1
q_2

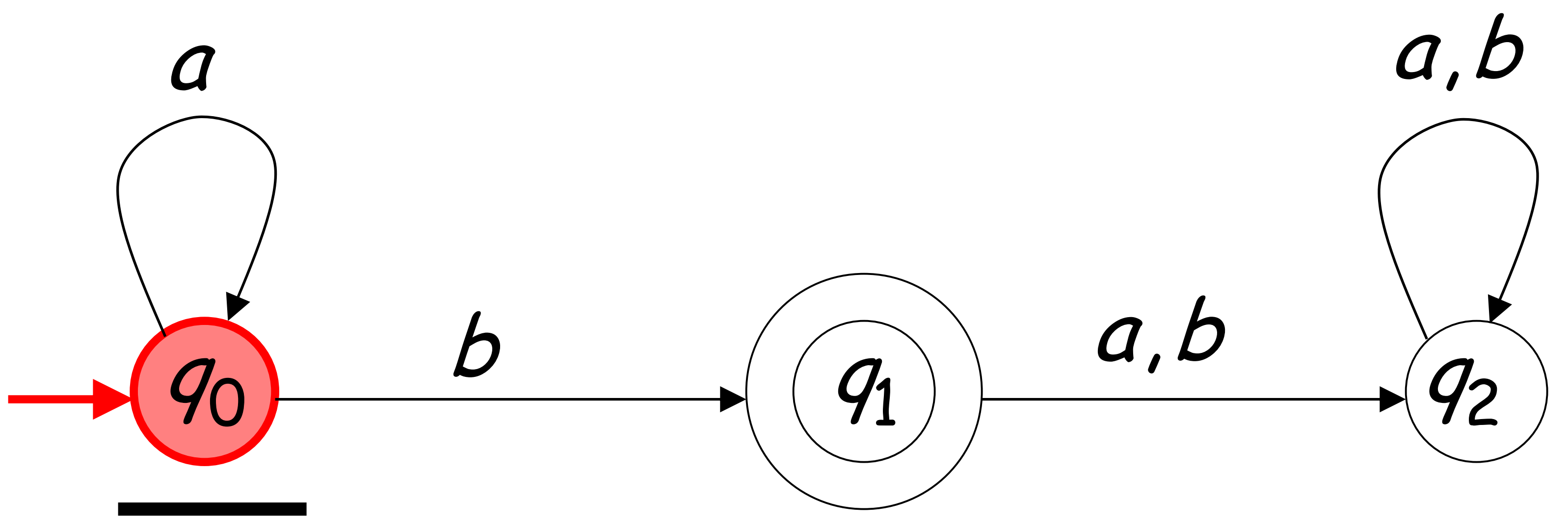
Input finished

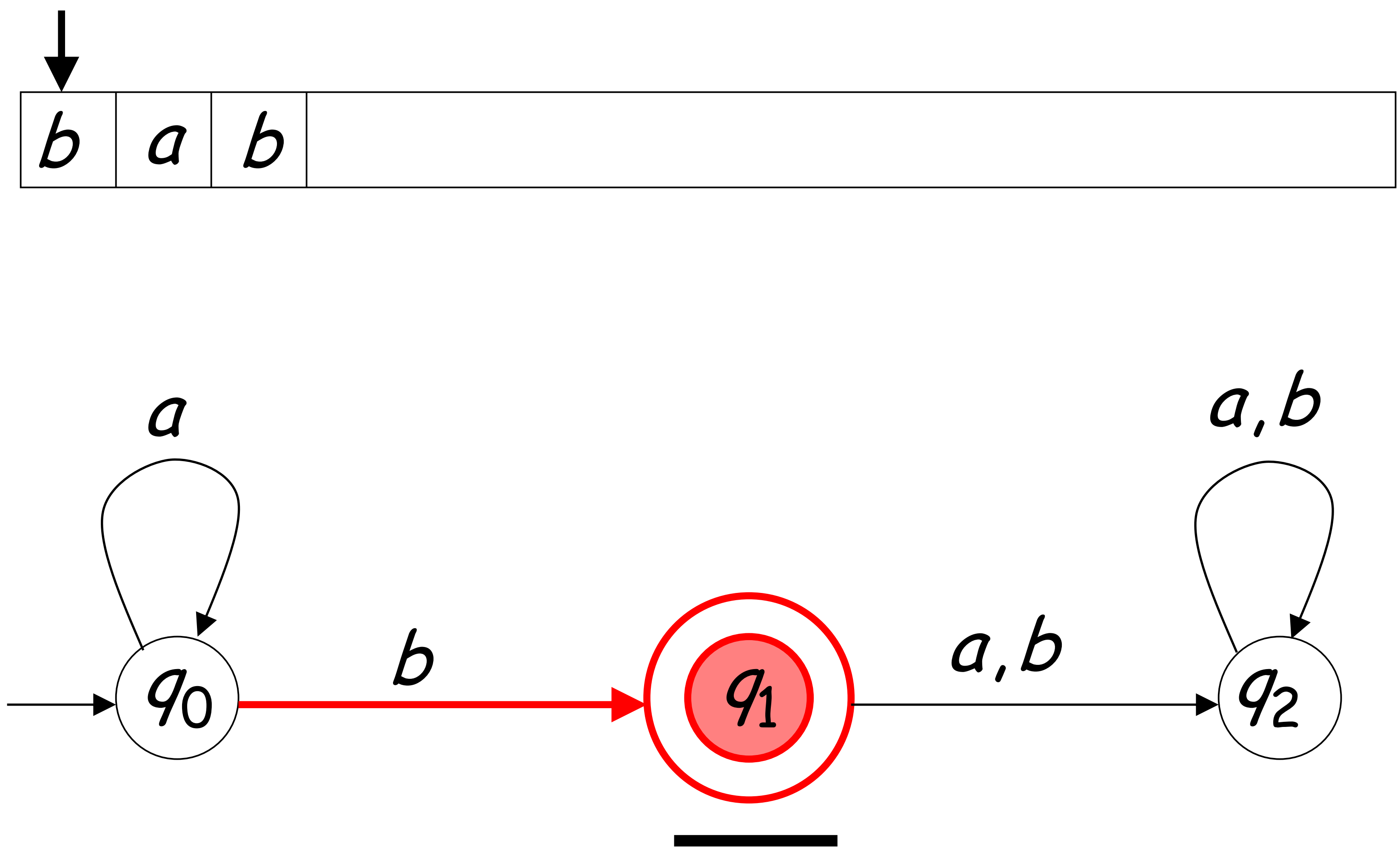


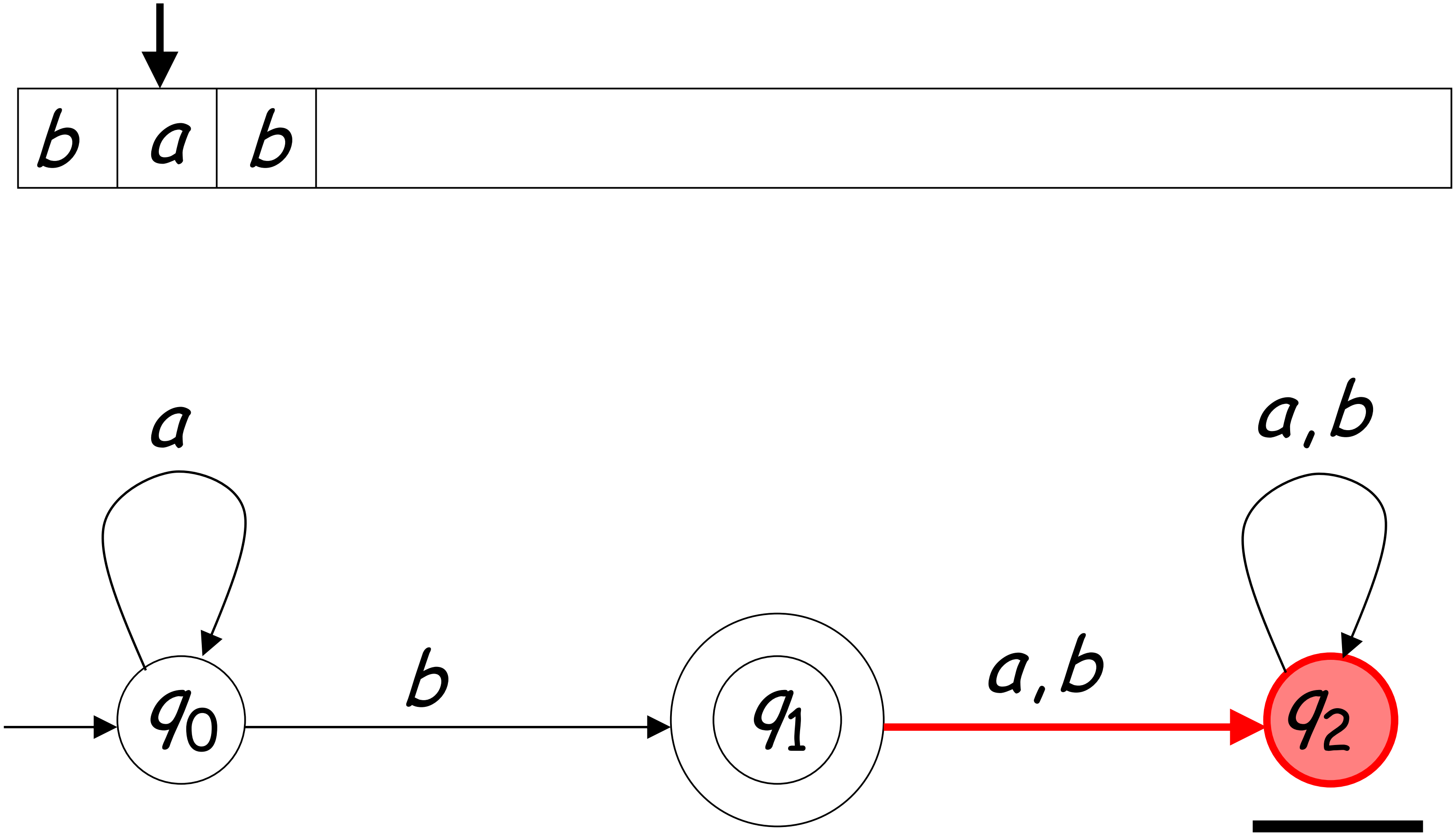
A rejection case



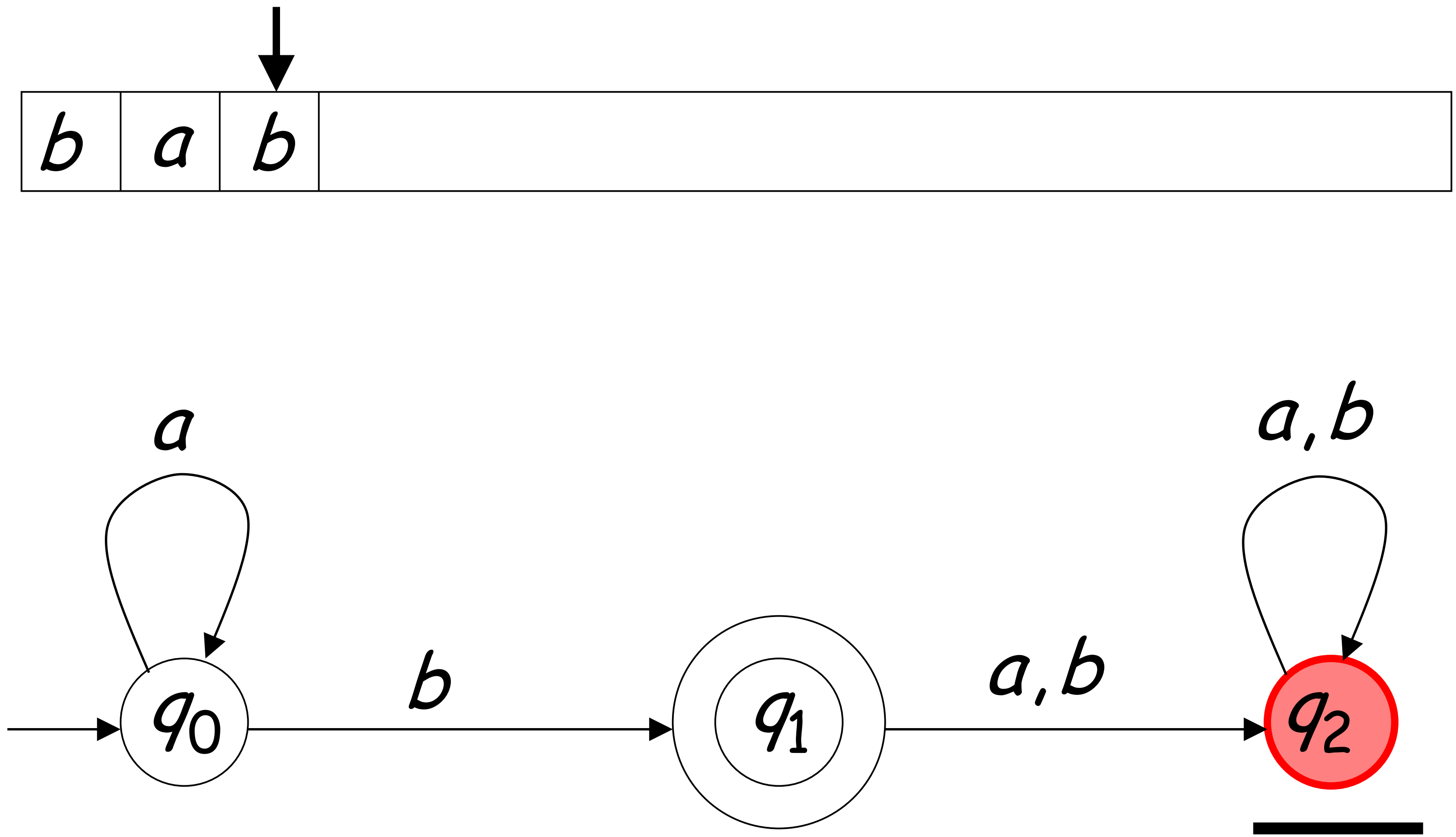
Input String




b a b
a
a,b
b
a,b
q_0
q_1
q_2

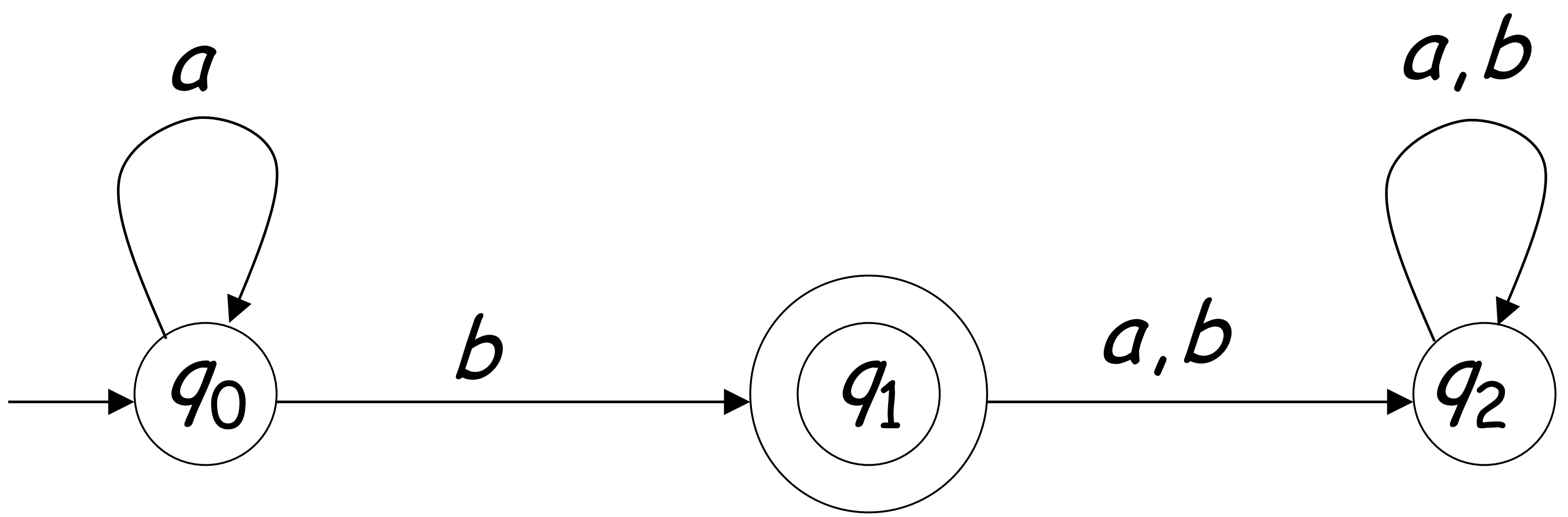

b
a
b
a
a,b
b
a,b
q_0
q_1
q_2

Input finished



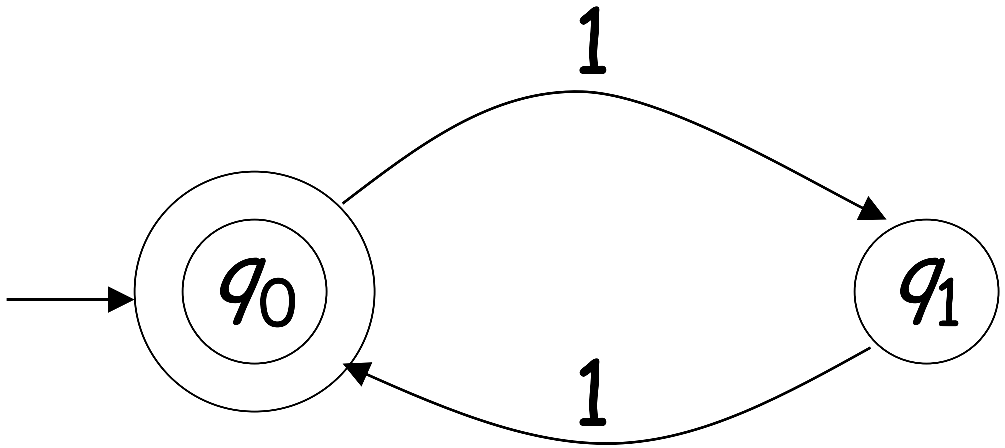
reject

Language Accepted: $L = \{a^n b : n \geq 0\}$



Another Example

Alphabet: $\Sigma = \{1\}$

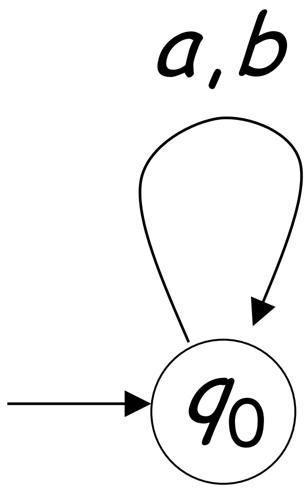


Language Accepted:

$$EVEN = \{x : x \in \Sigma^* \text{ and } x \text{ is even}\}$$
$$= \{\lambda, 11, 1111, 111111, \ldots\}$$

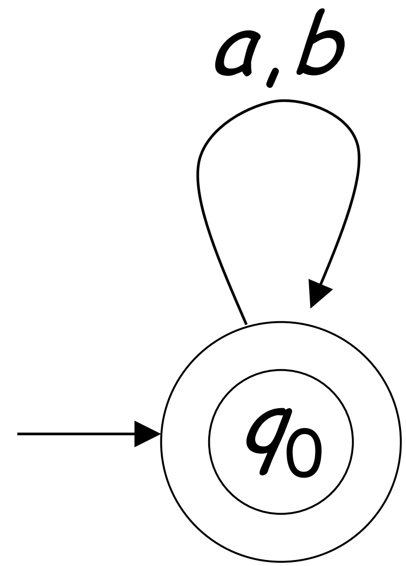
More DFA Examples

$$\Sigma = \{a, b\}$$



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

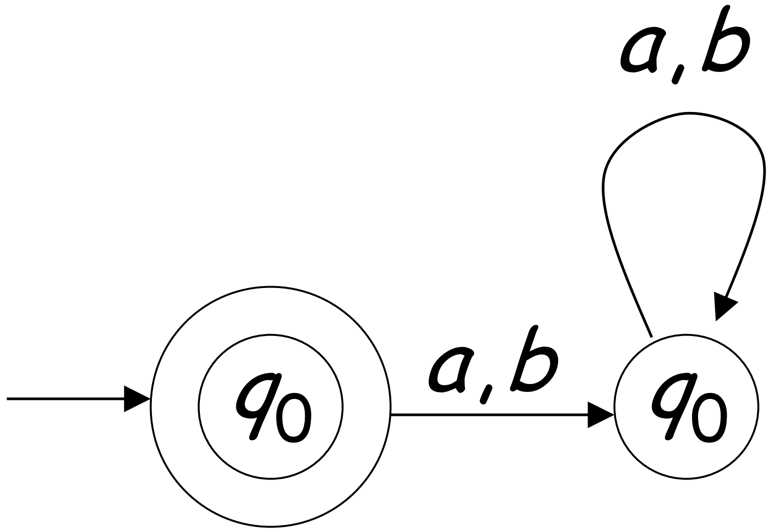
Empty language



$$L(M) = \Sigma^*$$

All strings

$$\Sigma = \{a, b\}$$

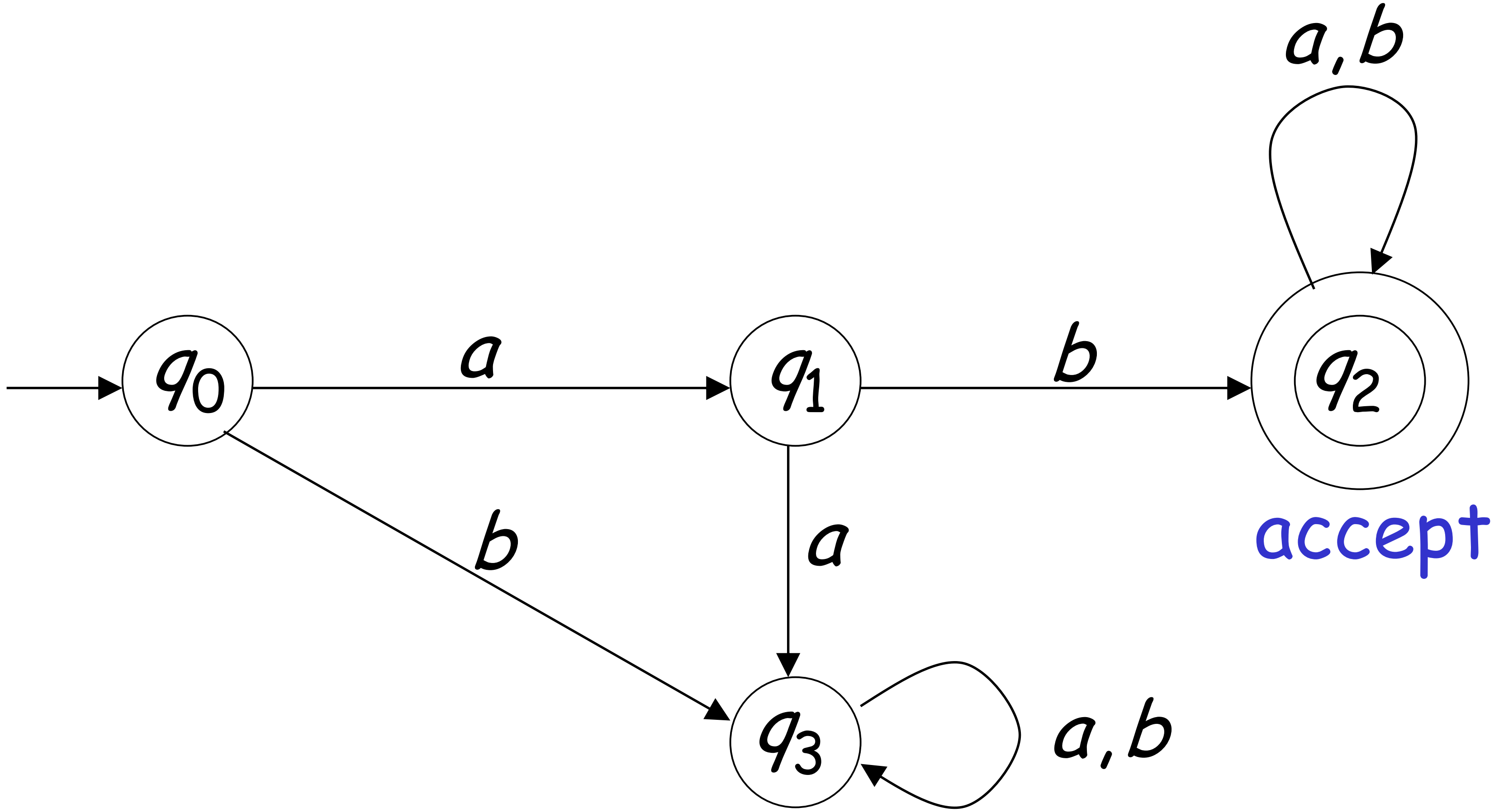


$$L(M) = \{\lambda\}$$

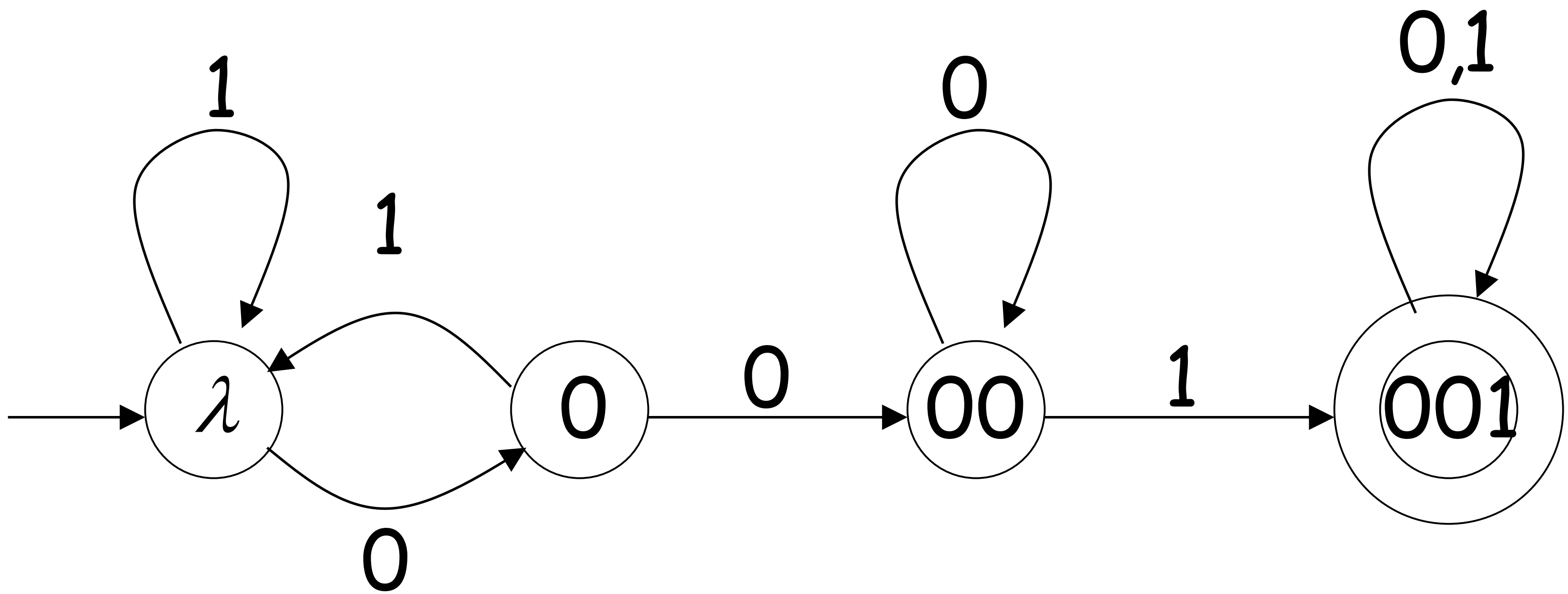
Language of the empty string

$$\Sigma = \{a, b\}$$

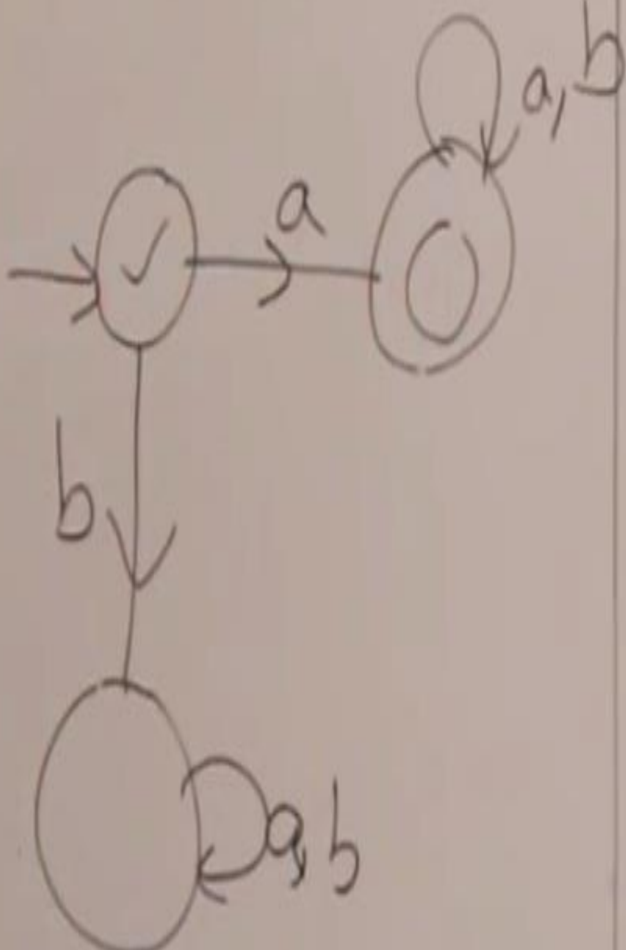
$L(M)$= { all strings with prefix ab }



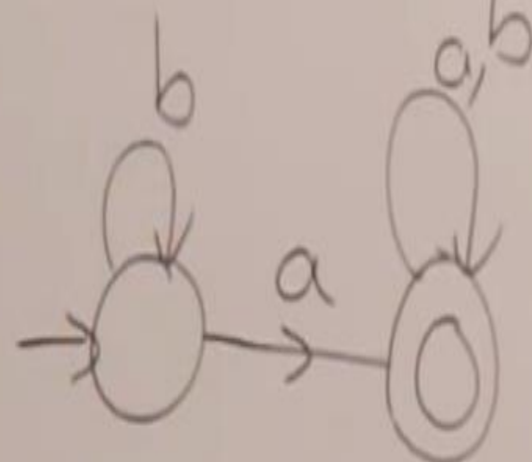
$L(M)$ = { all binary strings containing substring 001 }



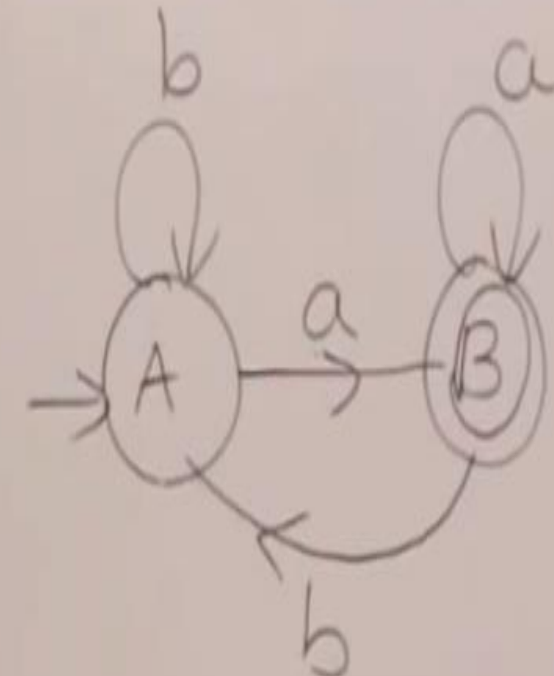
Starts with a



Containing a

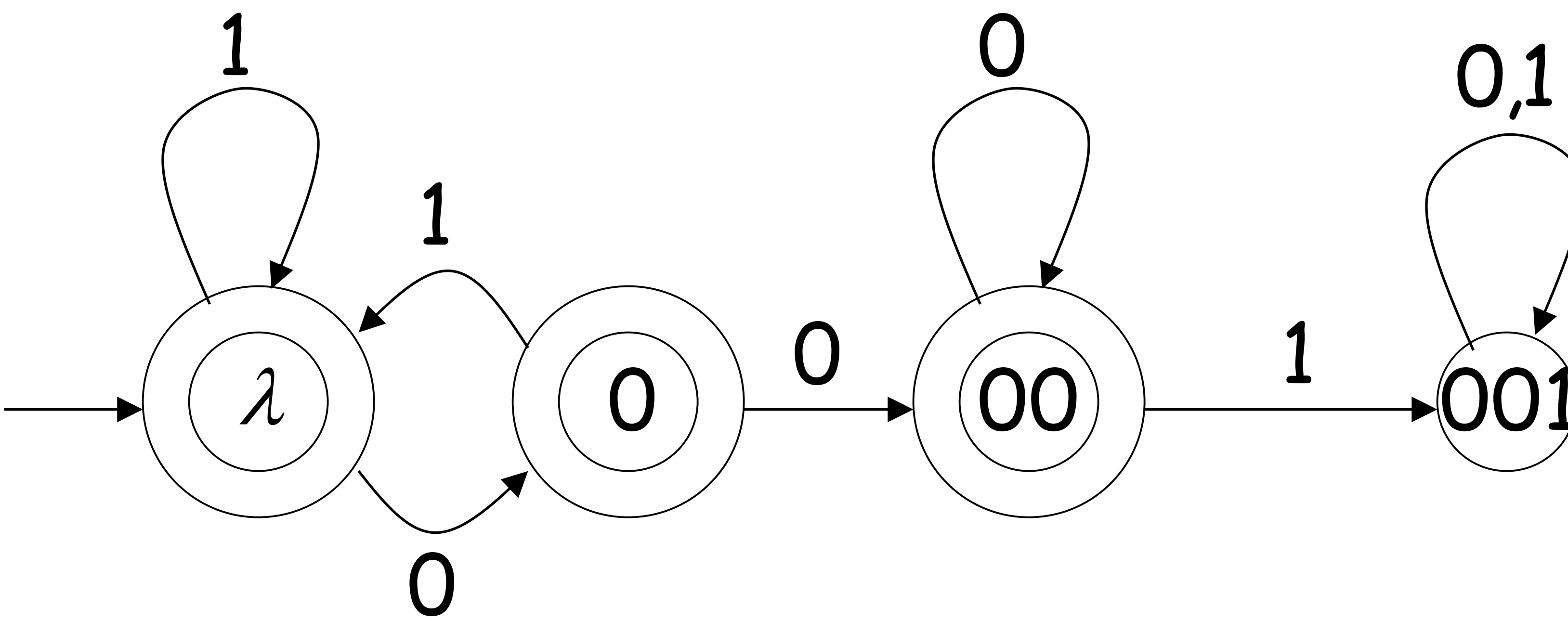


Ends with a



bbbab □ a

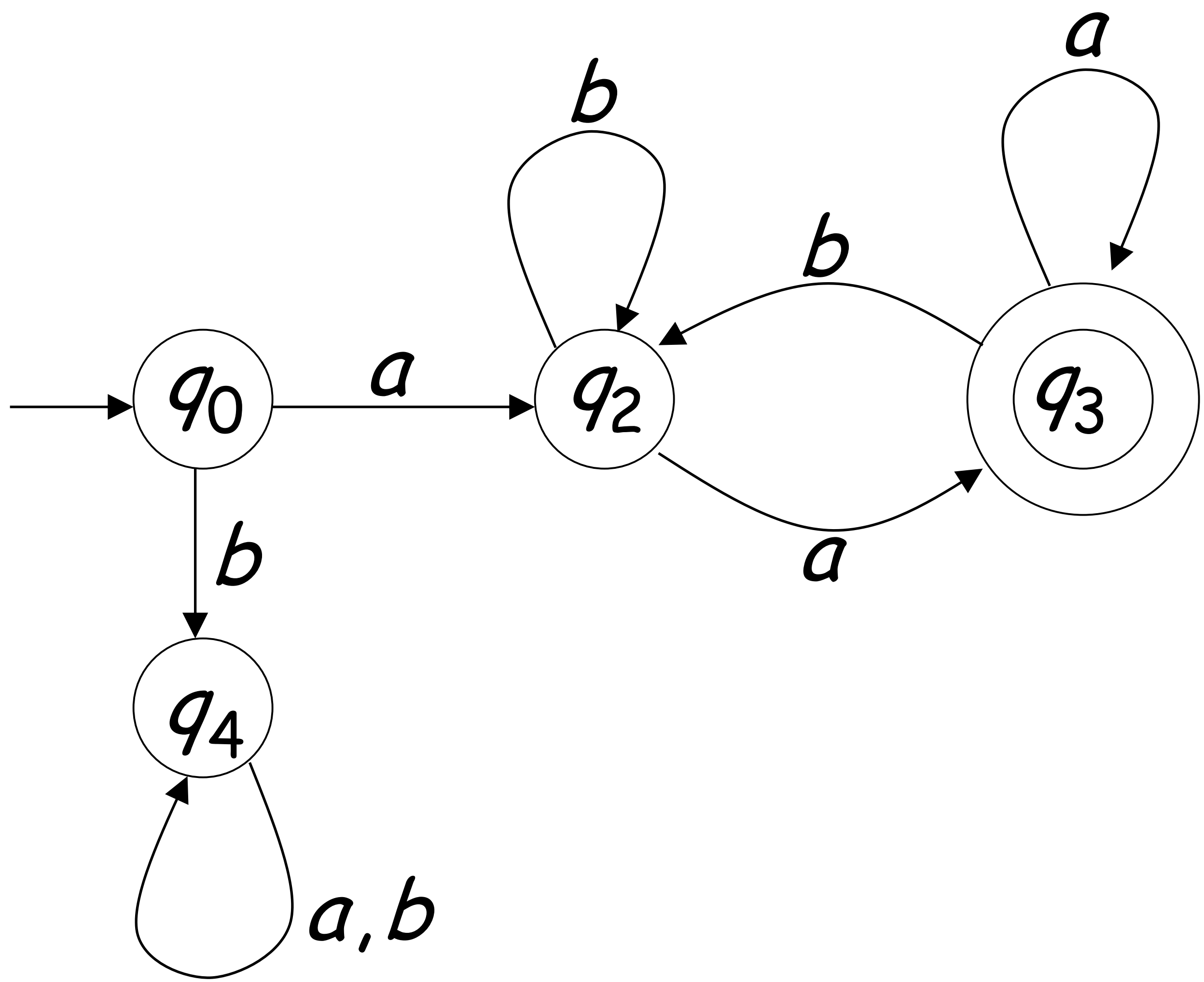
$L(M)$ = { all binary strings without substring 001 }



Review

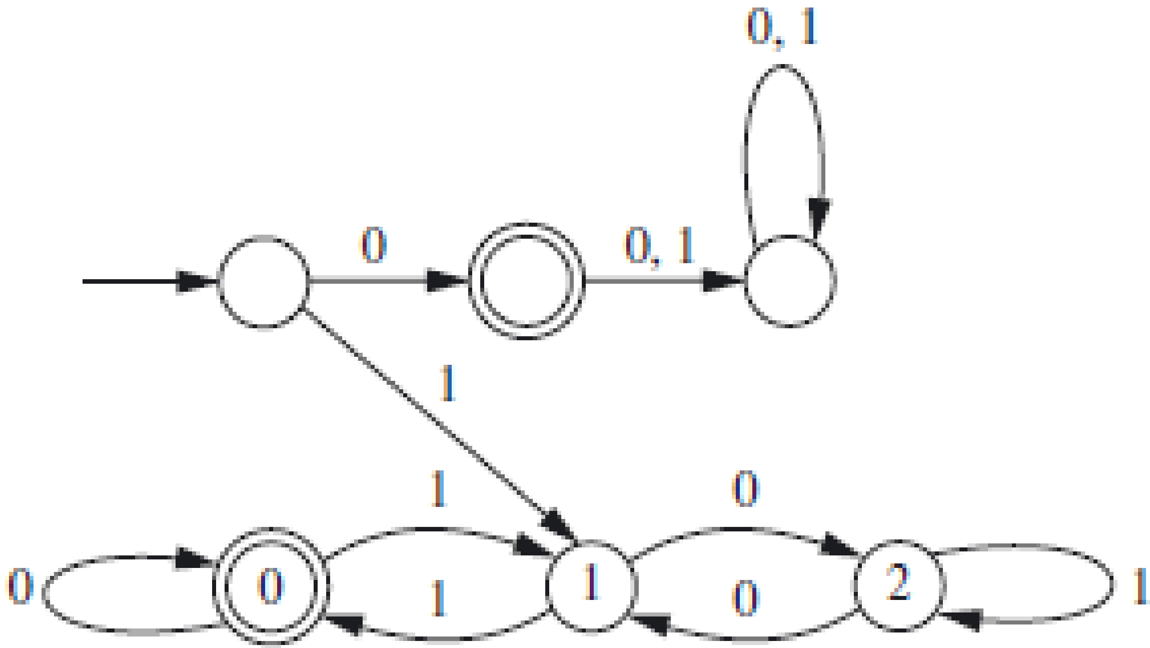
- DFA Examples

$$L(M) = \{awa : w \in \{a,b\}^*\}$$



Examples

- A Finite Automaton Accepting the Language of Strings Ending in aa over the alphabet {a,b}
- An FA accepting the strings ending with *b* *and not containing aa.*
- An FA Accepting Binary Representations of Integers Divisible by 3
- An FA Accepting Strings That Contain Either ab or bba.


0, 1
0
0, 1
1
1
0
0
0
1
0
1
2
1

Regular Languages

Definition:

A language L is regular if there is a DFA M that accepts it ($L(M) = L$)

The languages accepted by all DFAs form the family of regular languages

Example regular languages:

$\{abba\}$ $\{\lambda, ab, abba\}$

$\{a^n b : n \geq 0\}$ $\{awa : w \in \{a,b\}^*\}$

{ all strings in {a,b}* with prefix ab }

{ all binary strings without substring 001}

$\{x : x \in \{1\}^*$ and x is even$\}$

$\{\ \}$ $\{\lambda\}$ $\{a,b\}^*$

There exist automata that accept these languages (see previous slides).

There exist languages which are not Regular:

$$L = \{a^n b^n : n \geq 0\}$$

$$ADDITION = \{x + y = z \ : x = 1^n, y = 1^m, z = 1^k, \\ n + m = k\}$$

There is no DFA that accepts these languages

(we will prove this in a later class)

Review

- DFA Examples
- Regular Language

A language L is regular if it is recognized by a deterministic finite automaton (DFA), i.e. if there is a DFA M such that L = L (M).

L = { w | w contains 001} is regular

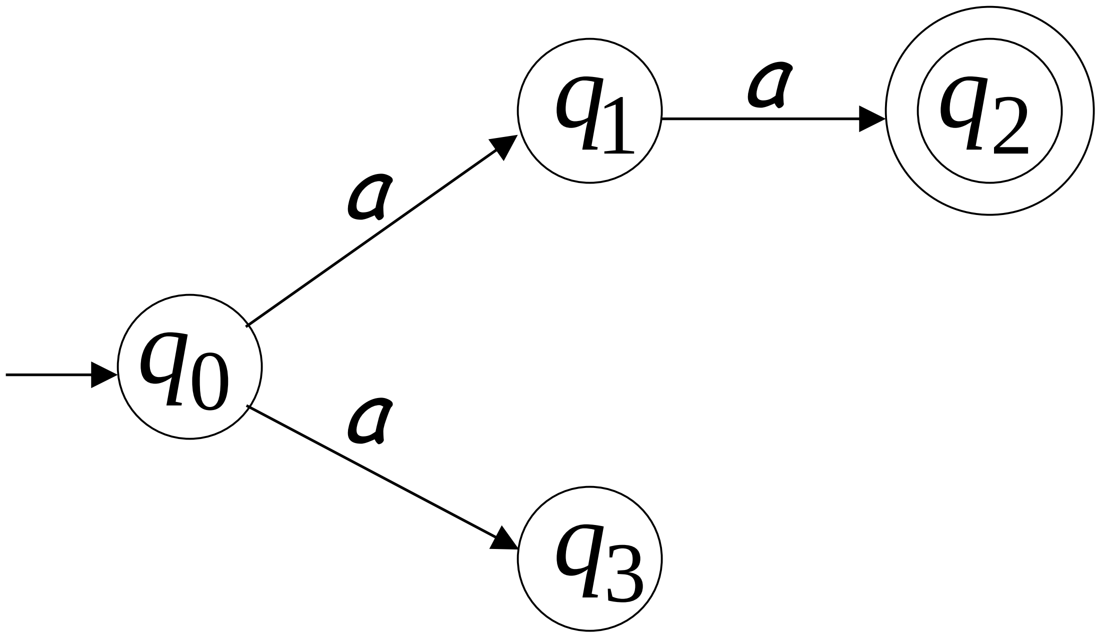
L = { w | w has an even number of 1’s} is regular

- FA which read string made of {0,1} and accepts those string which end with 00 or 11.(Insem-2017, 4 Marks)
- Design FA Accepting Language of Strings Ending in b and not containing the substring aa.
- An FA accepting Binary Representations of Integers Divisible by 3.

Non-Deterministic Finite Automata

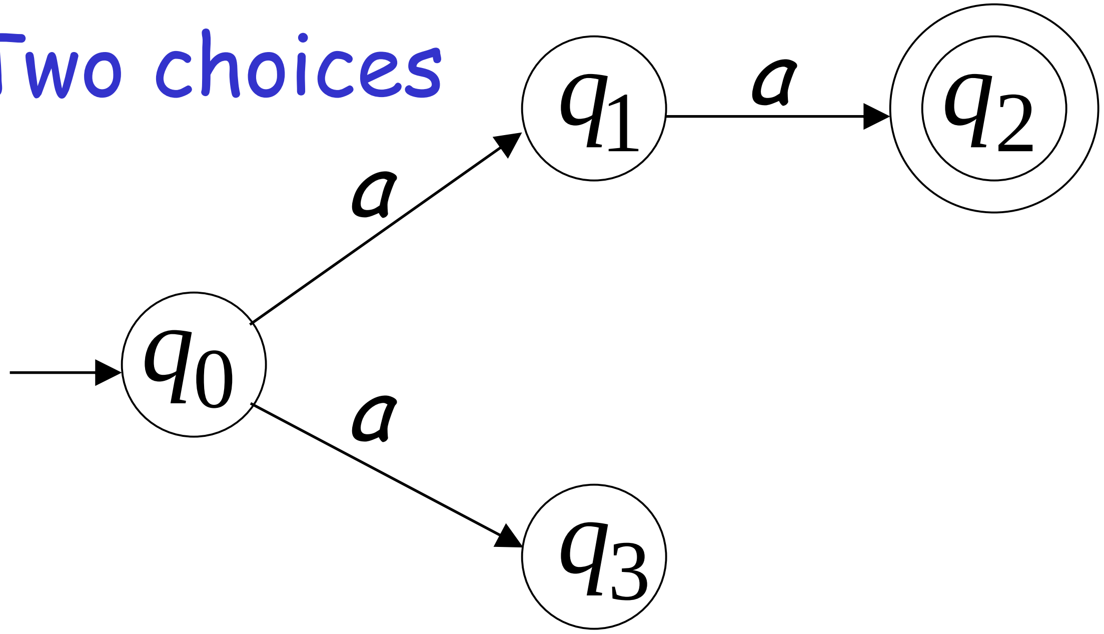
Nondeterministic Finite Automaton (NFA)

Alphabet = $\{a\}$

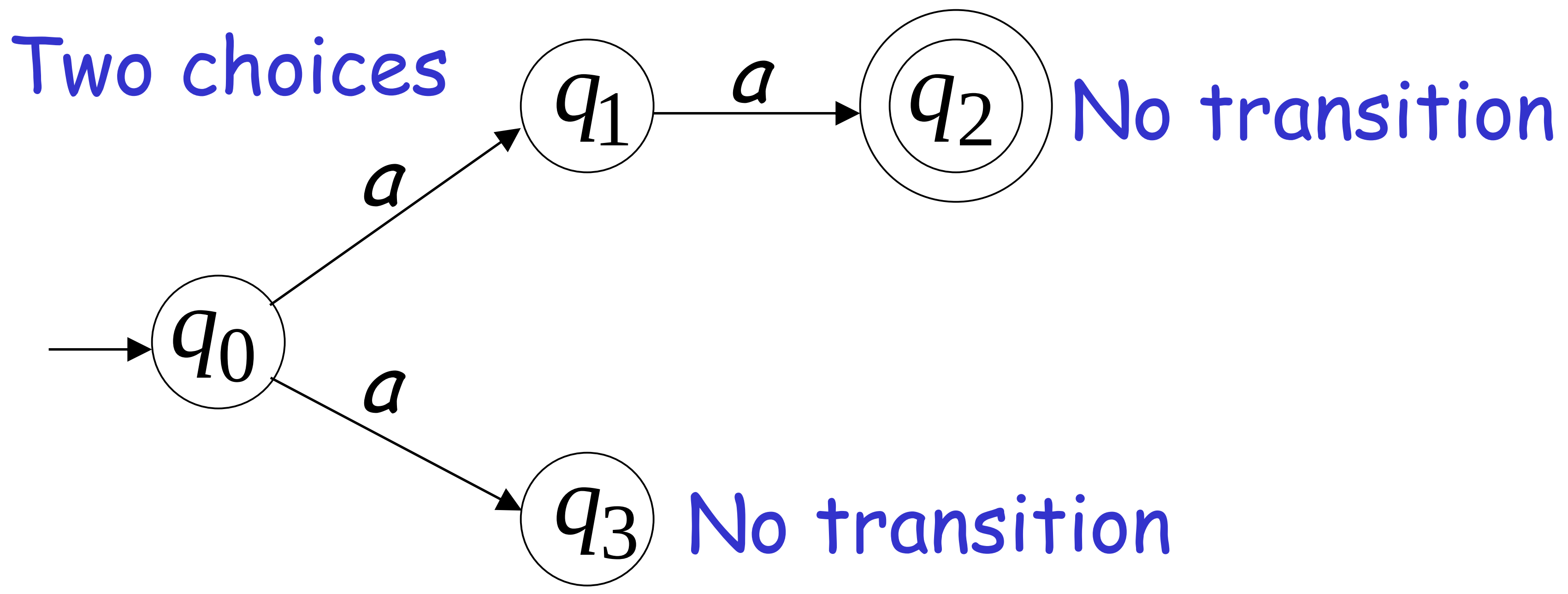


Alphabet = $\{a\}$

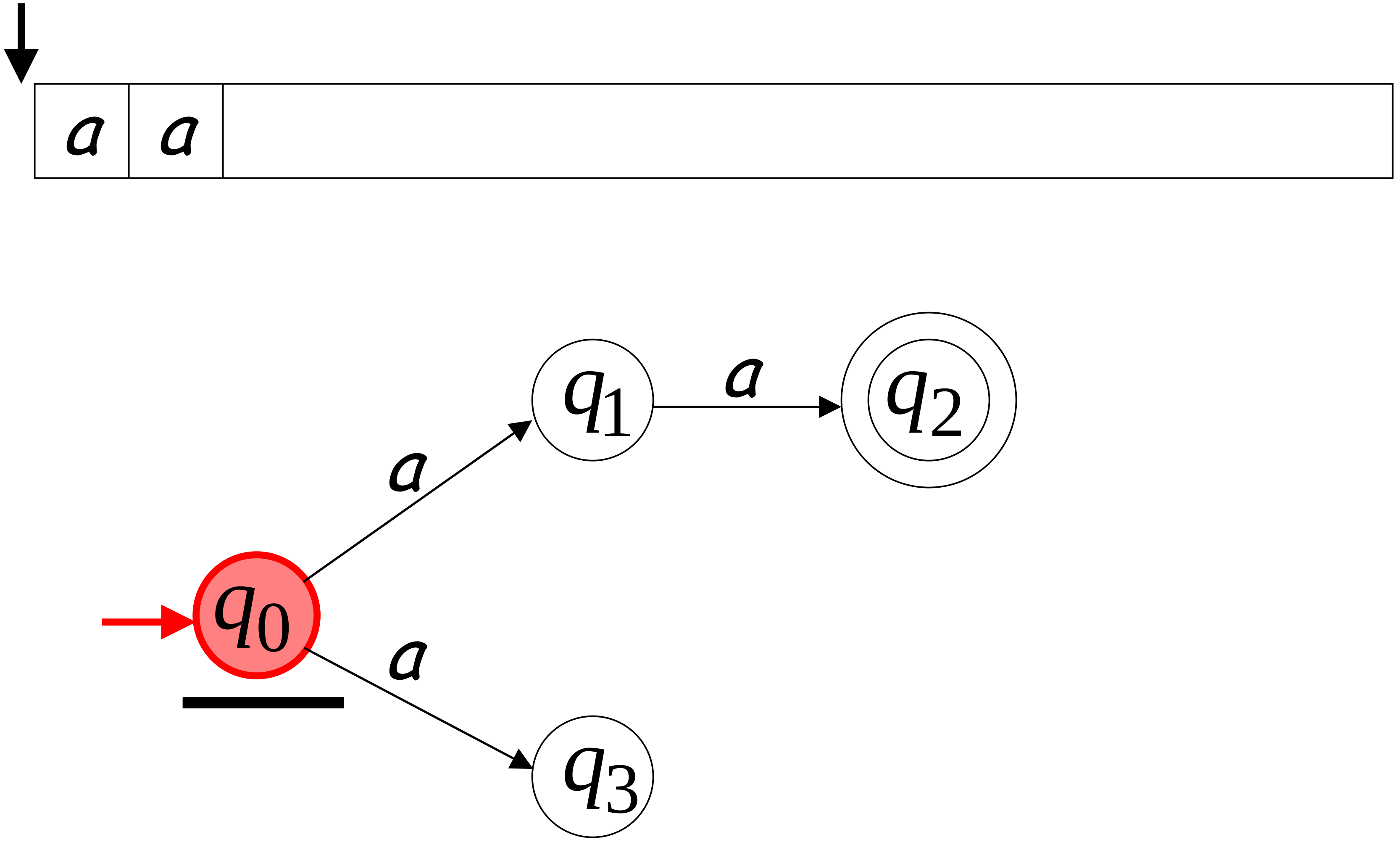
Two choices



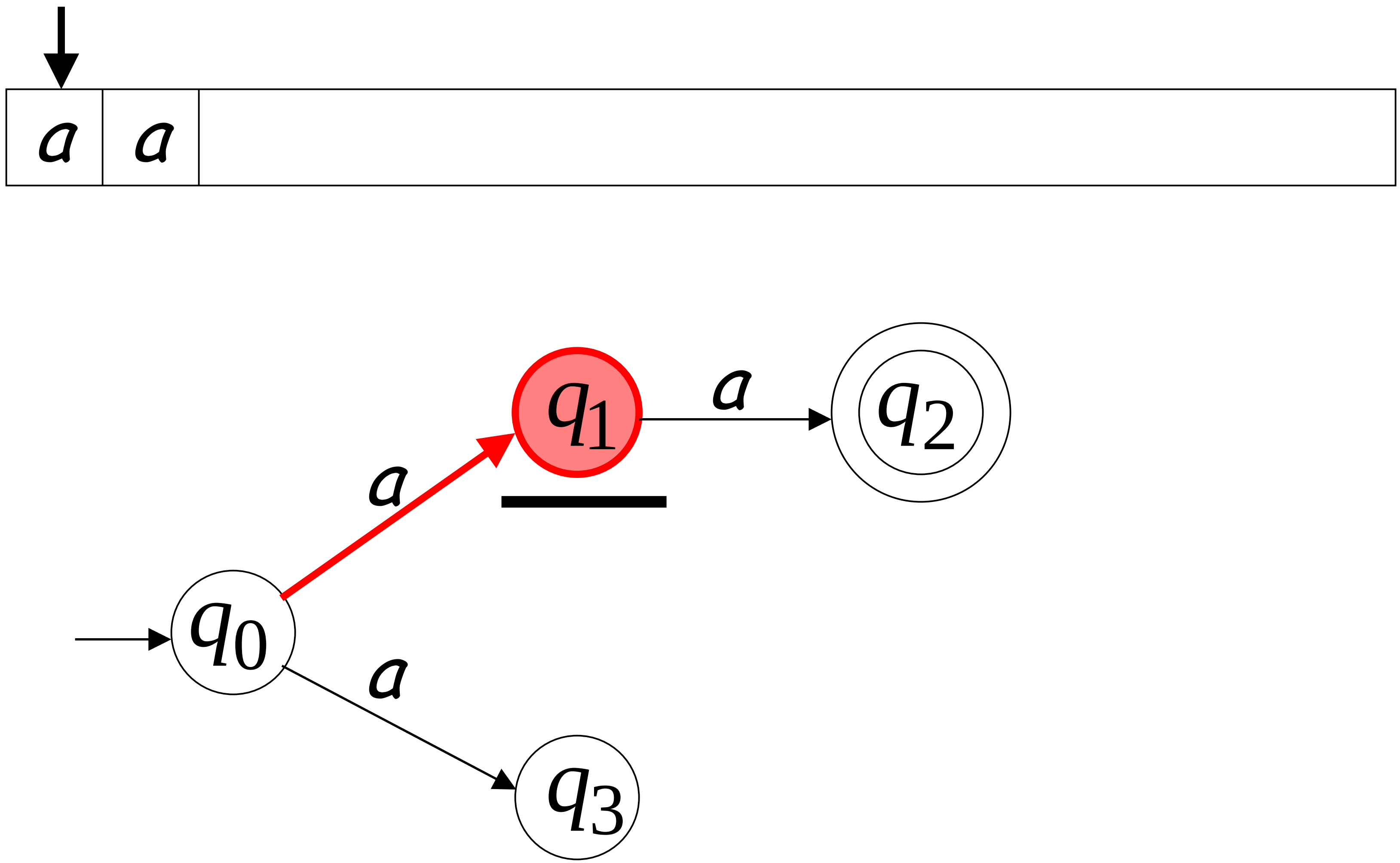
Alphabet = $\{a\}$



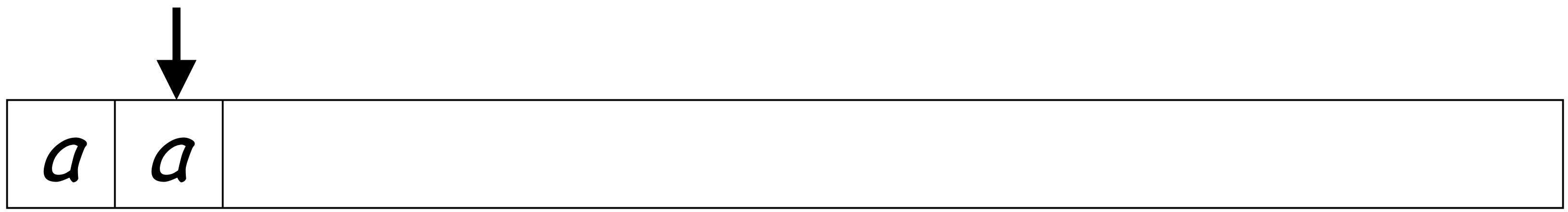
First Choice



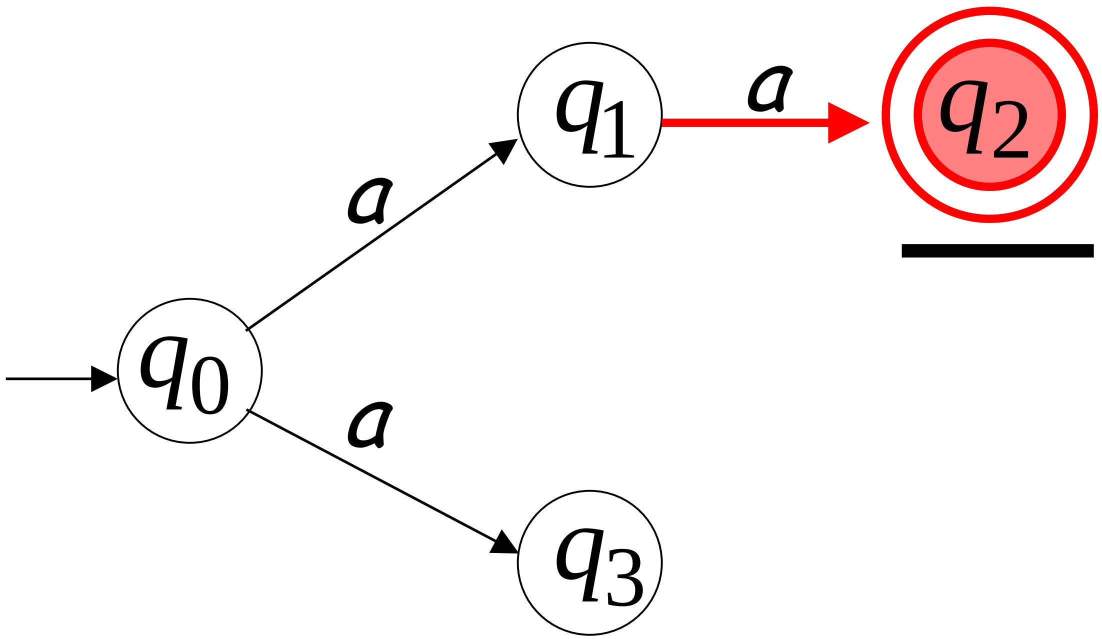
First Choice



First Choice

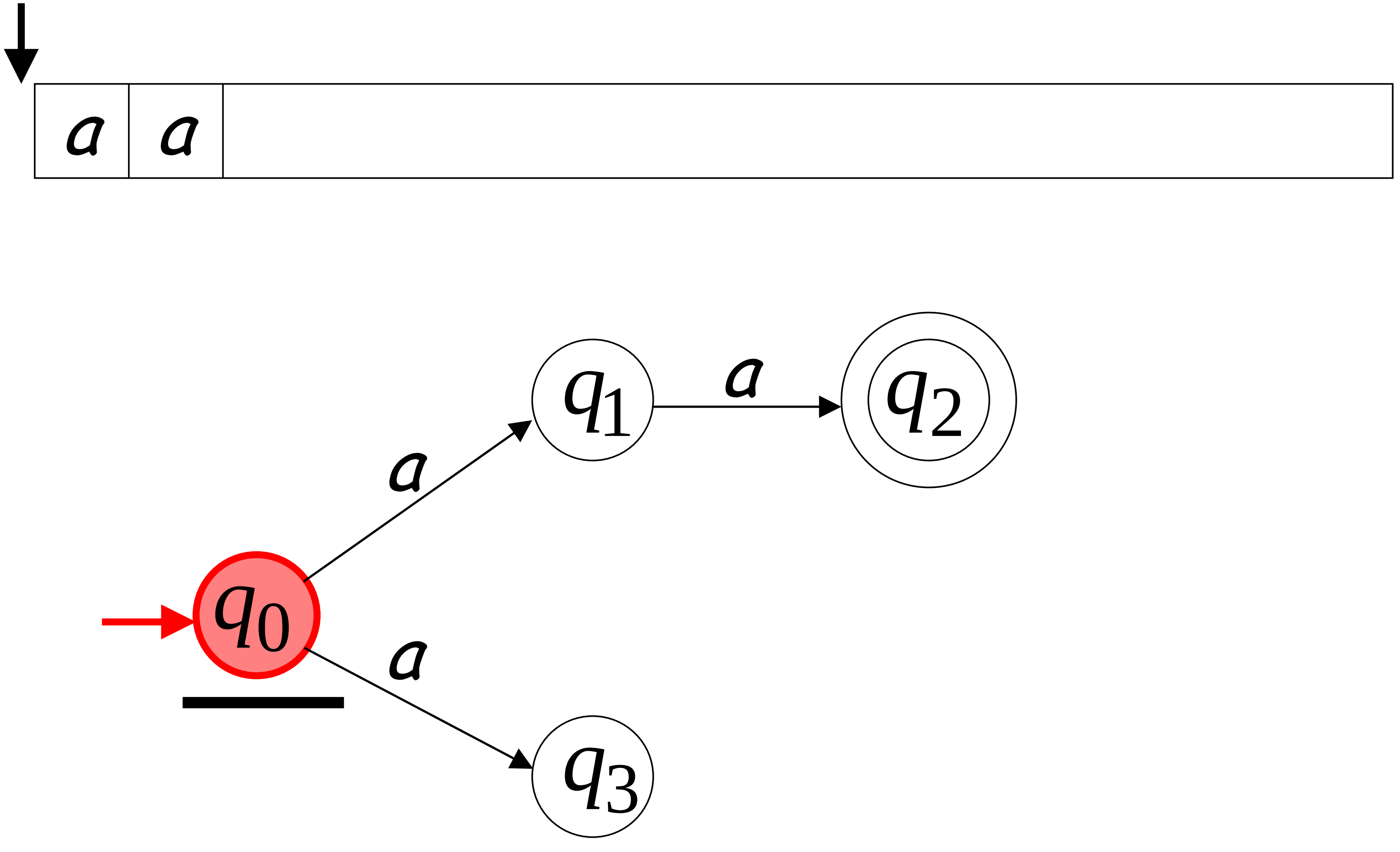


All input is consumed

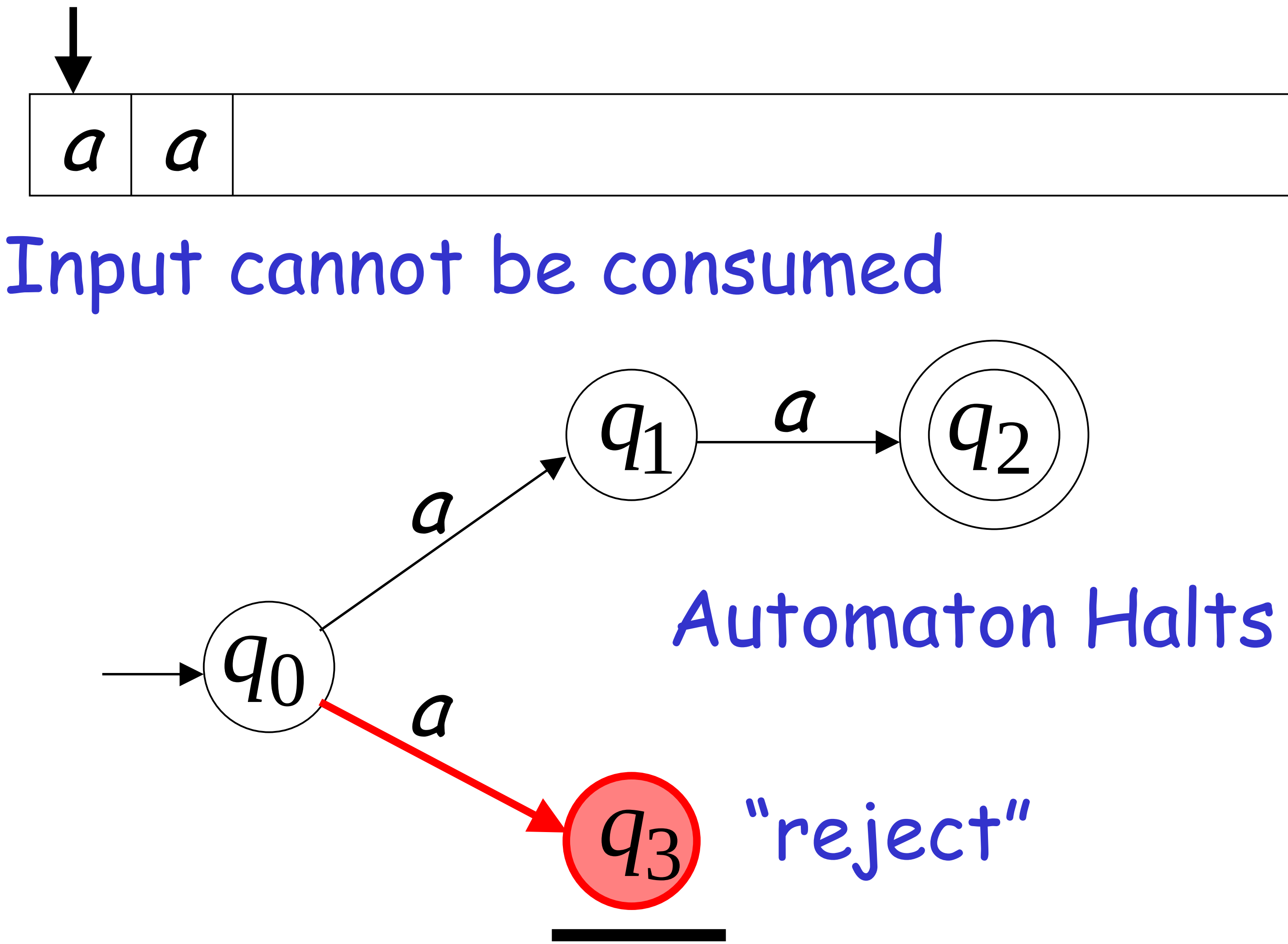


"accept"

Second Choice



Second Choice



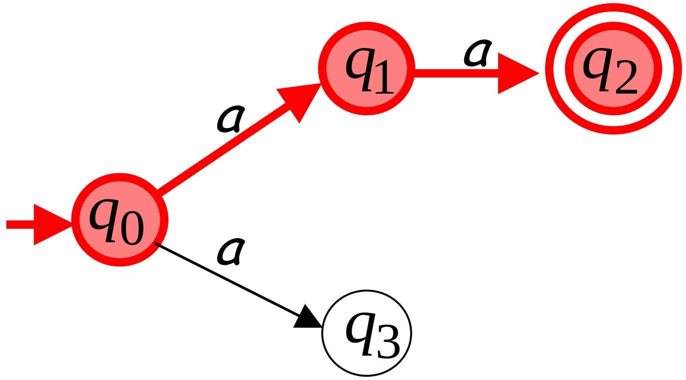
An NFA accepts a string:

if there is a computation of the NFA that accepts the string

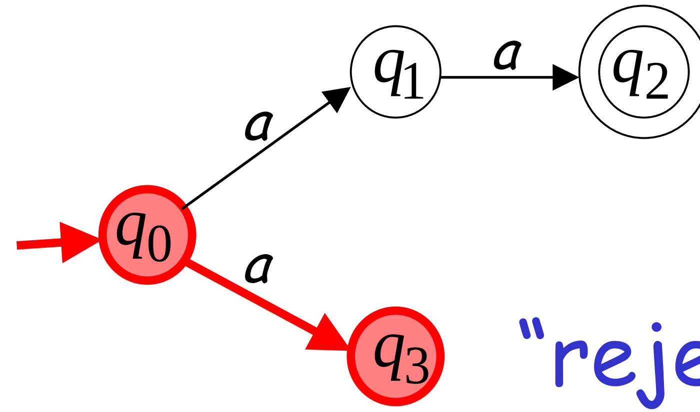
i.e., all the input string is processed and the automaton is in an accepting state

aa is accepted by the NFA:

"accept"



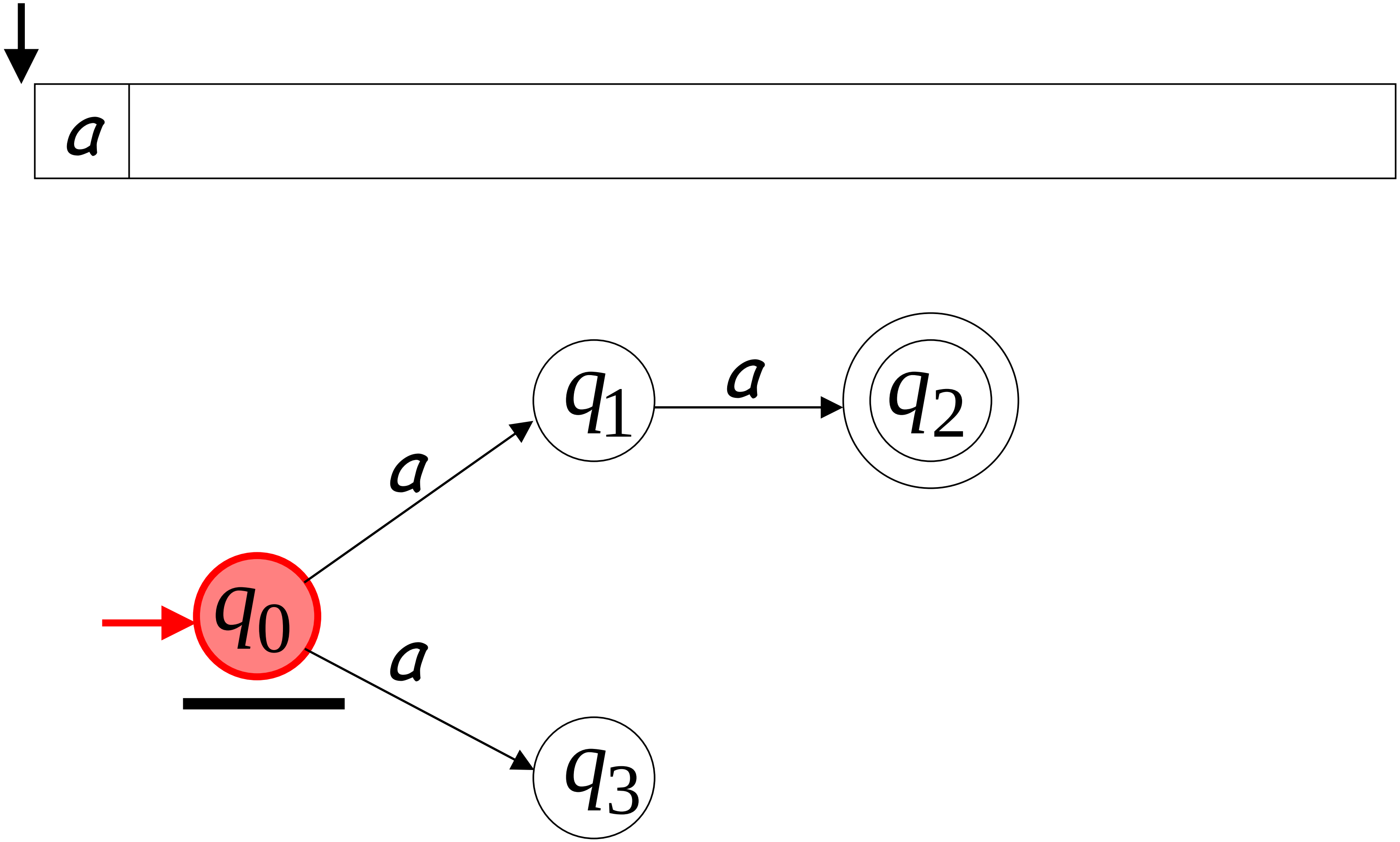
because this computation accepts aa



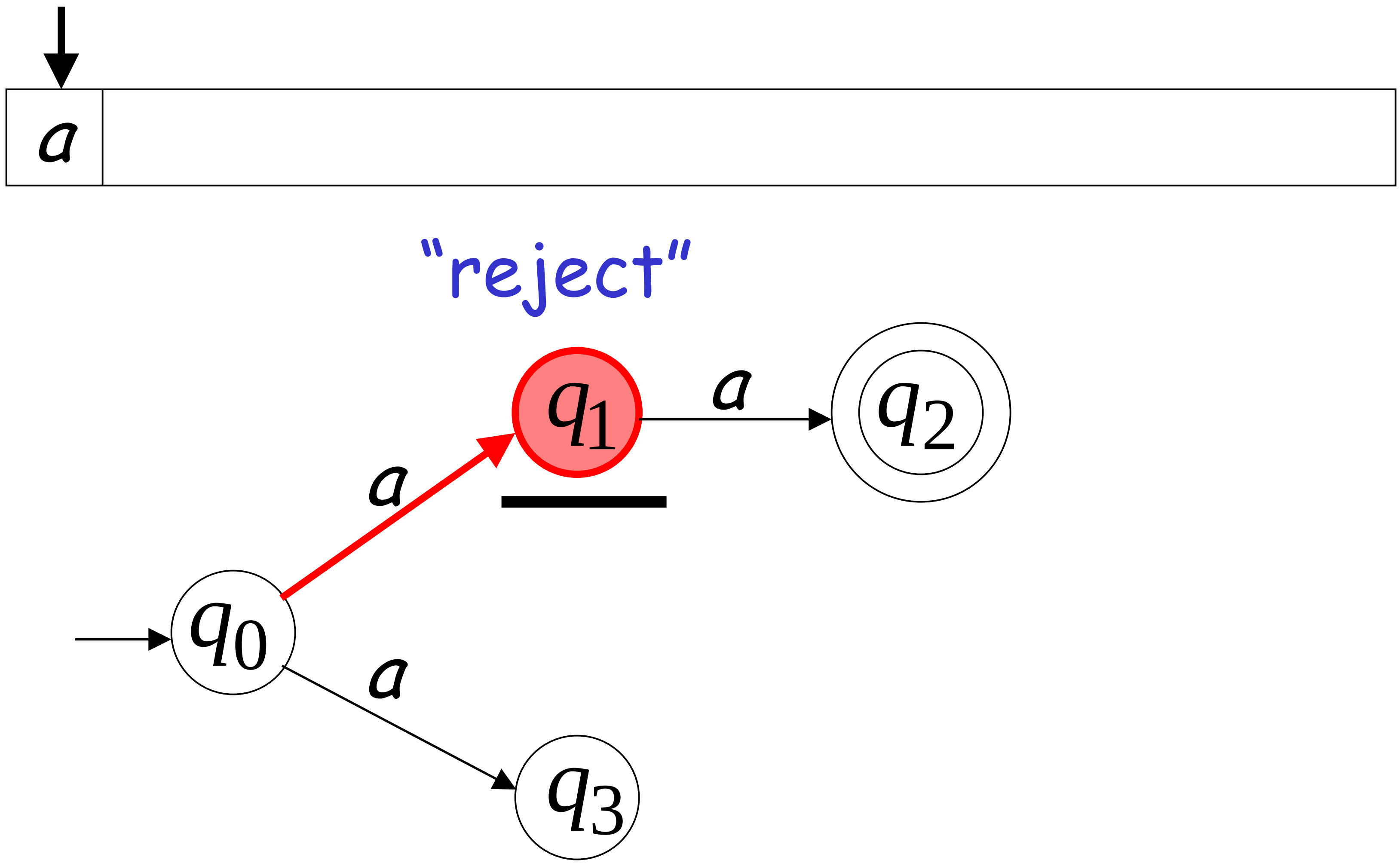
"reject"

this computation is ignored

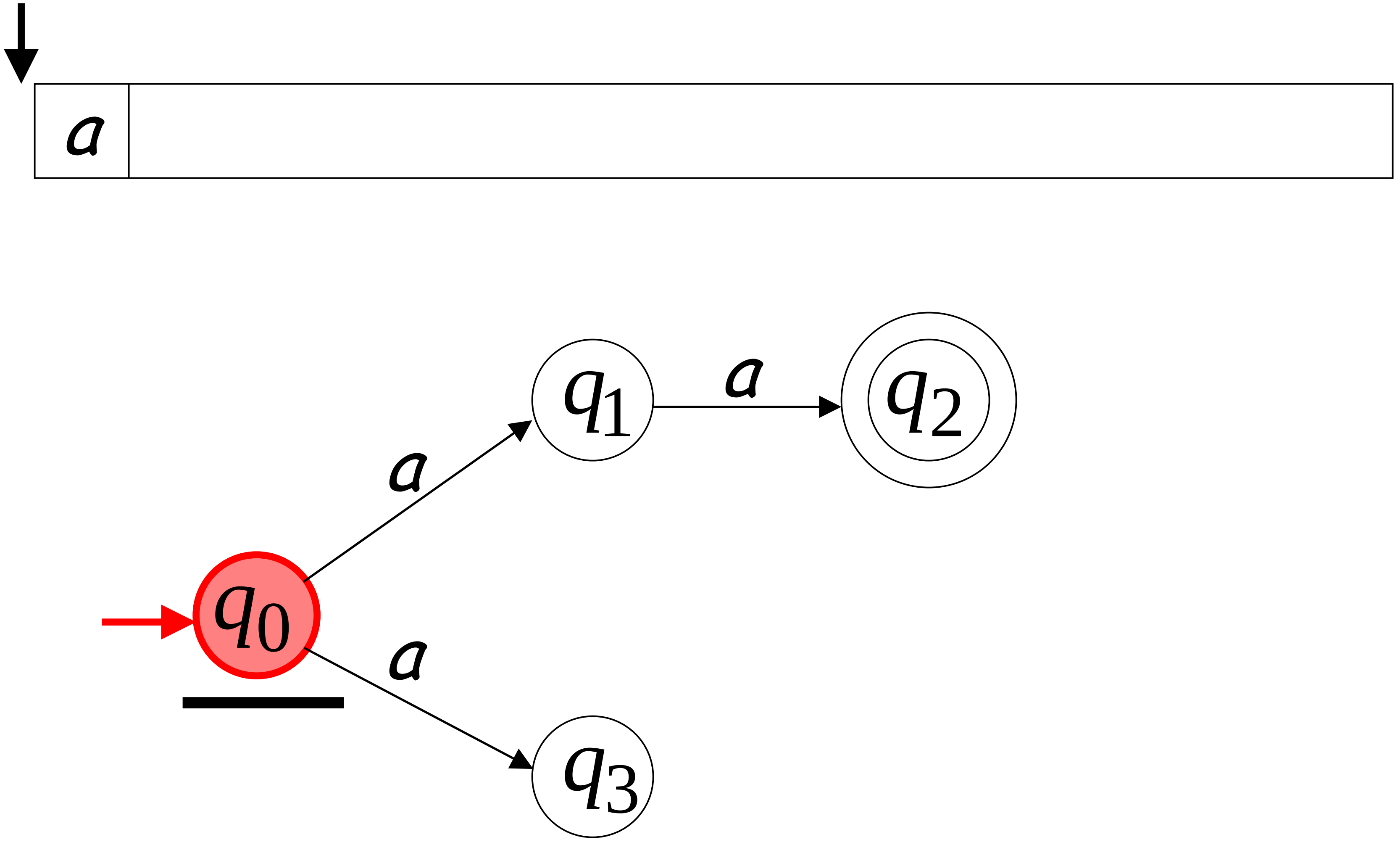
Rejection example



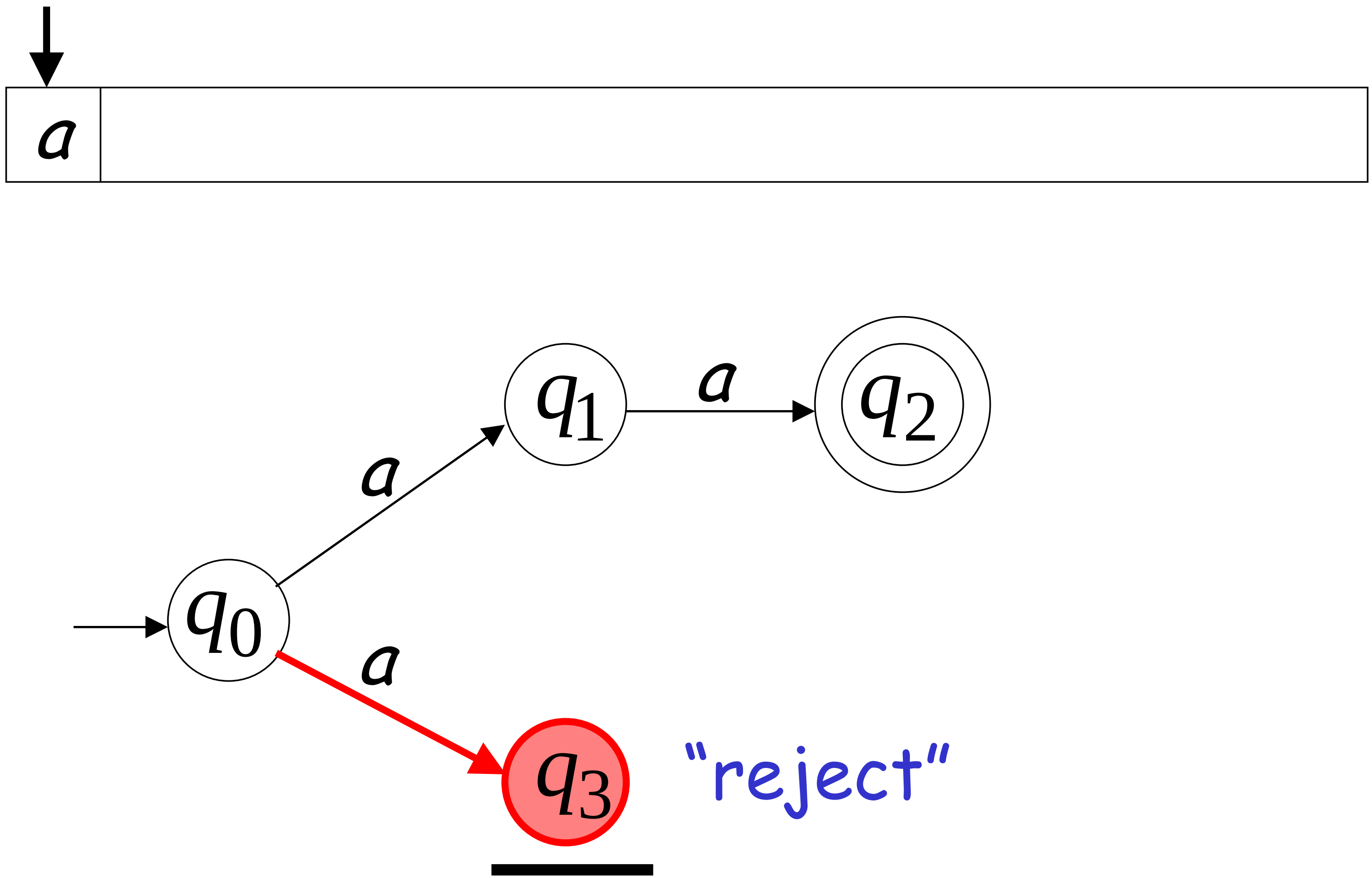
First Choice



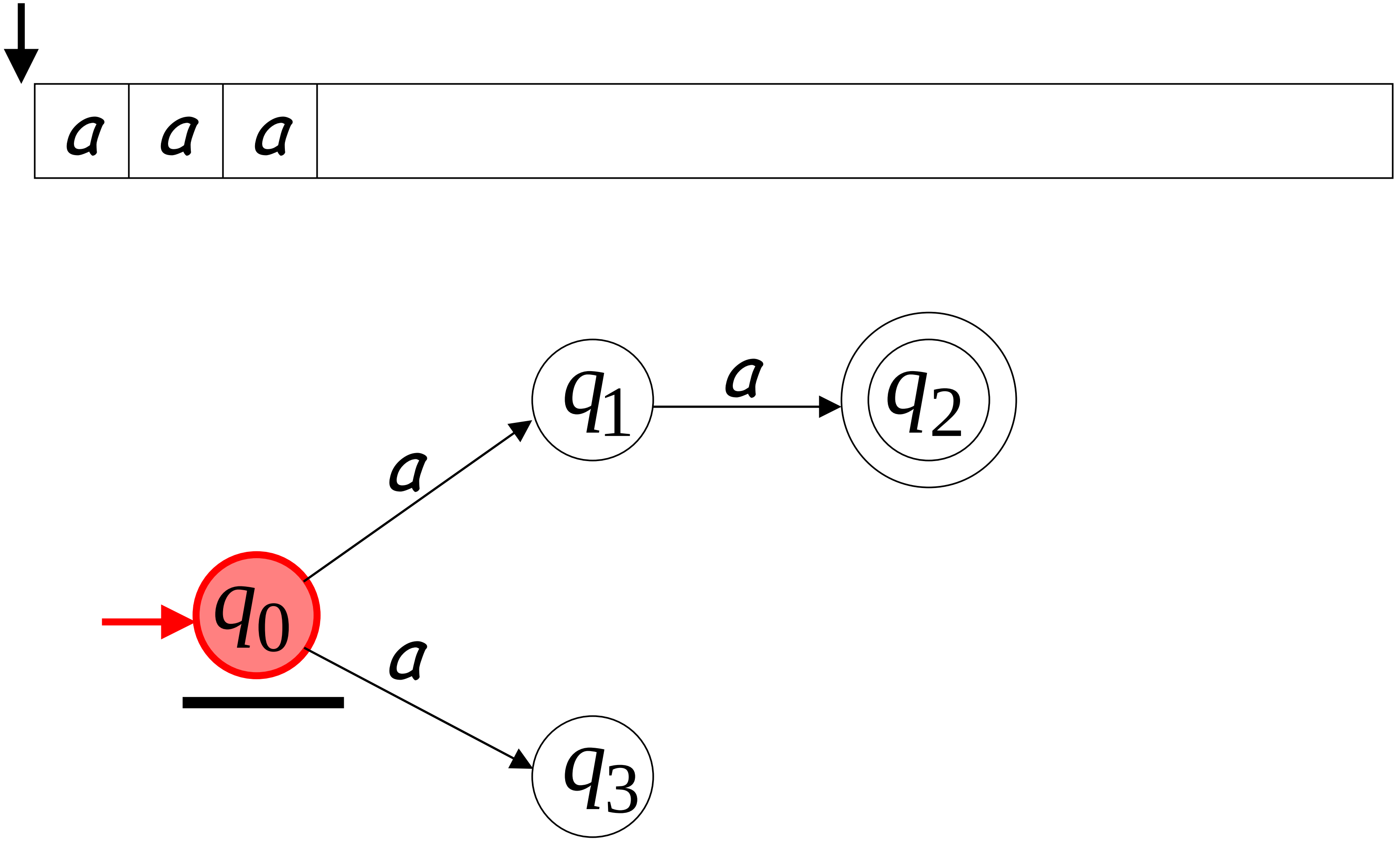
Second Choice



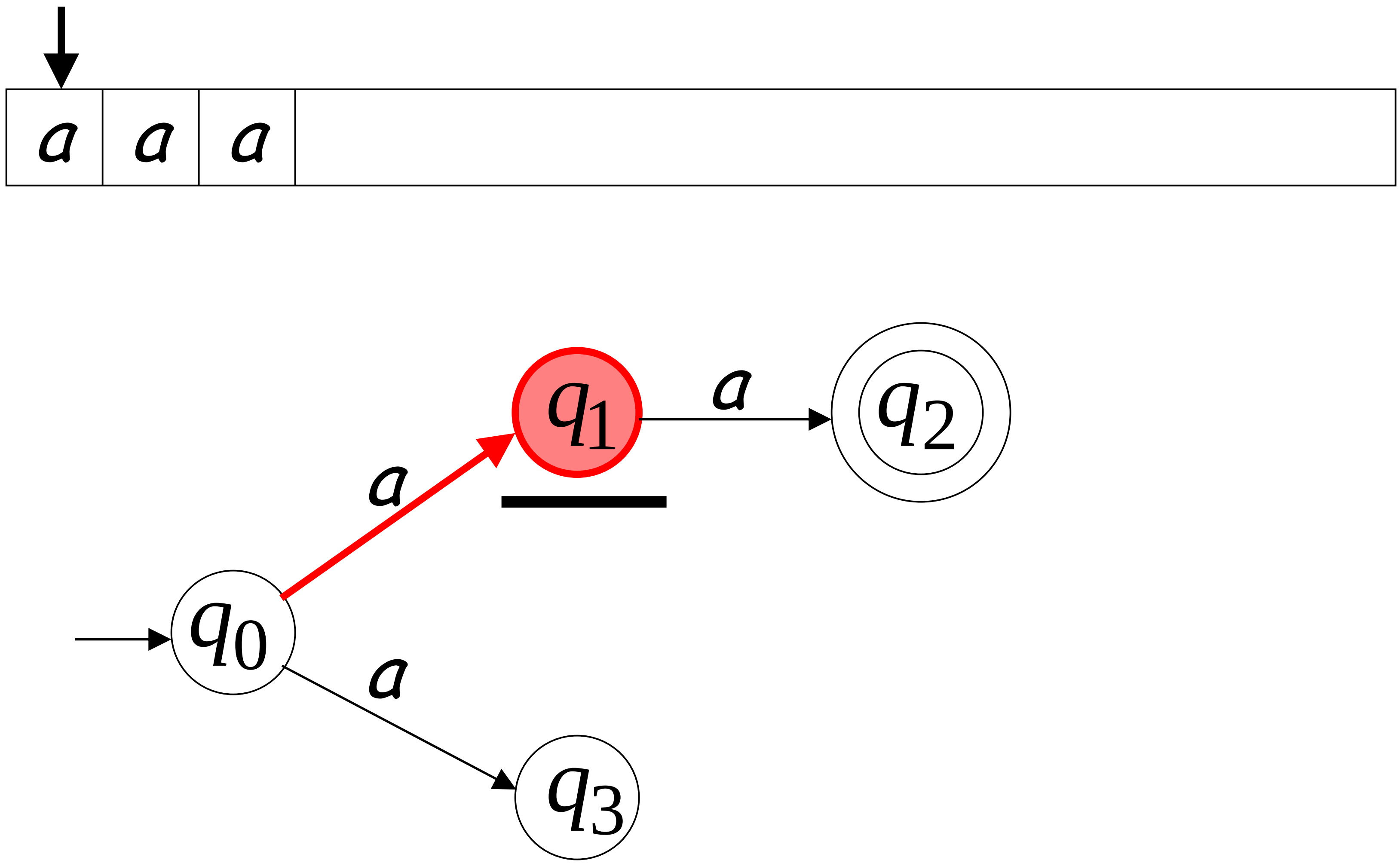
Second Choice



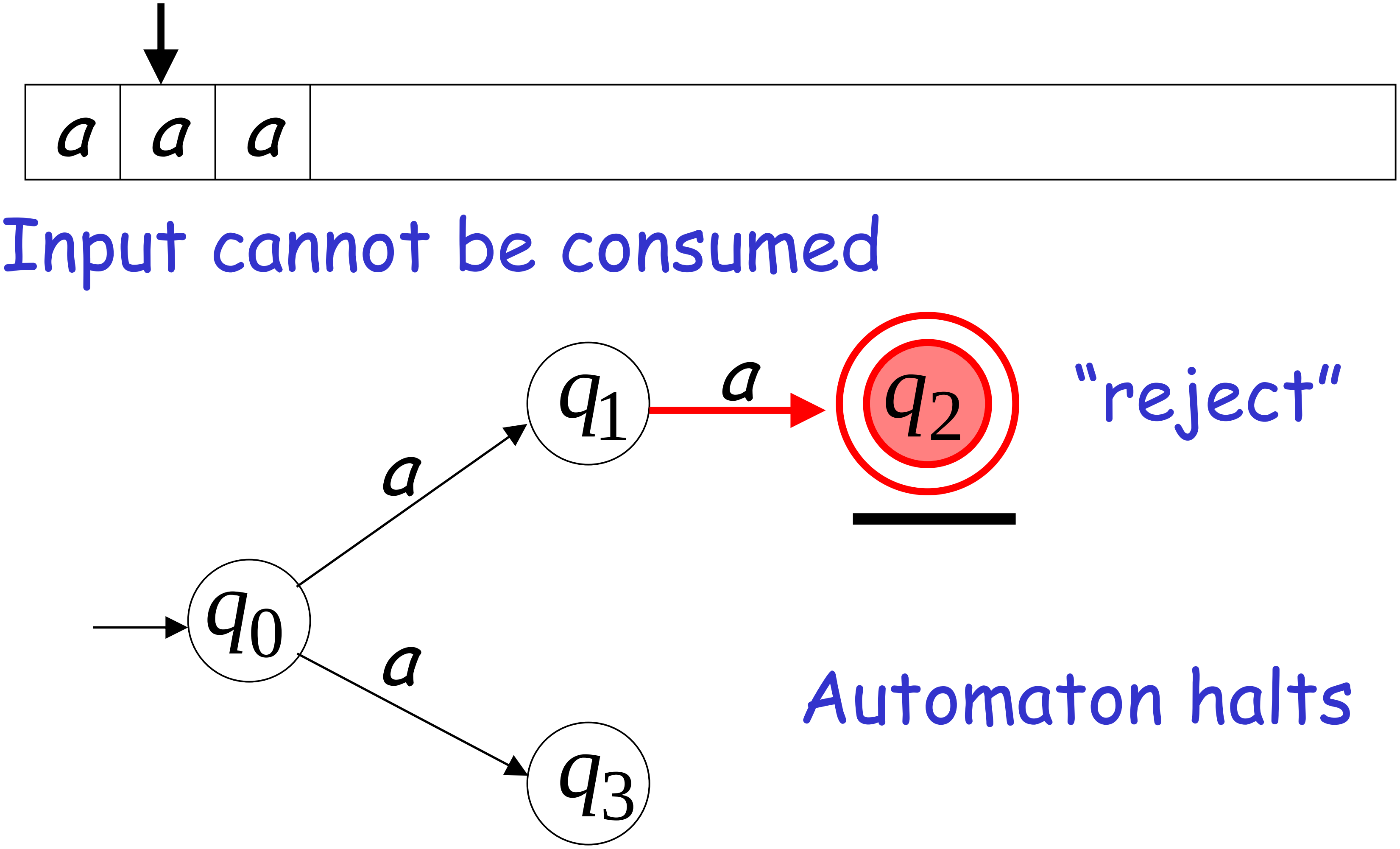
Another Rejection example



First Choice



First Choice

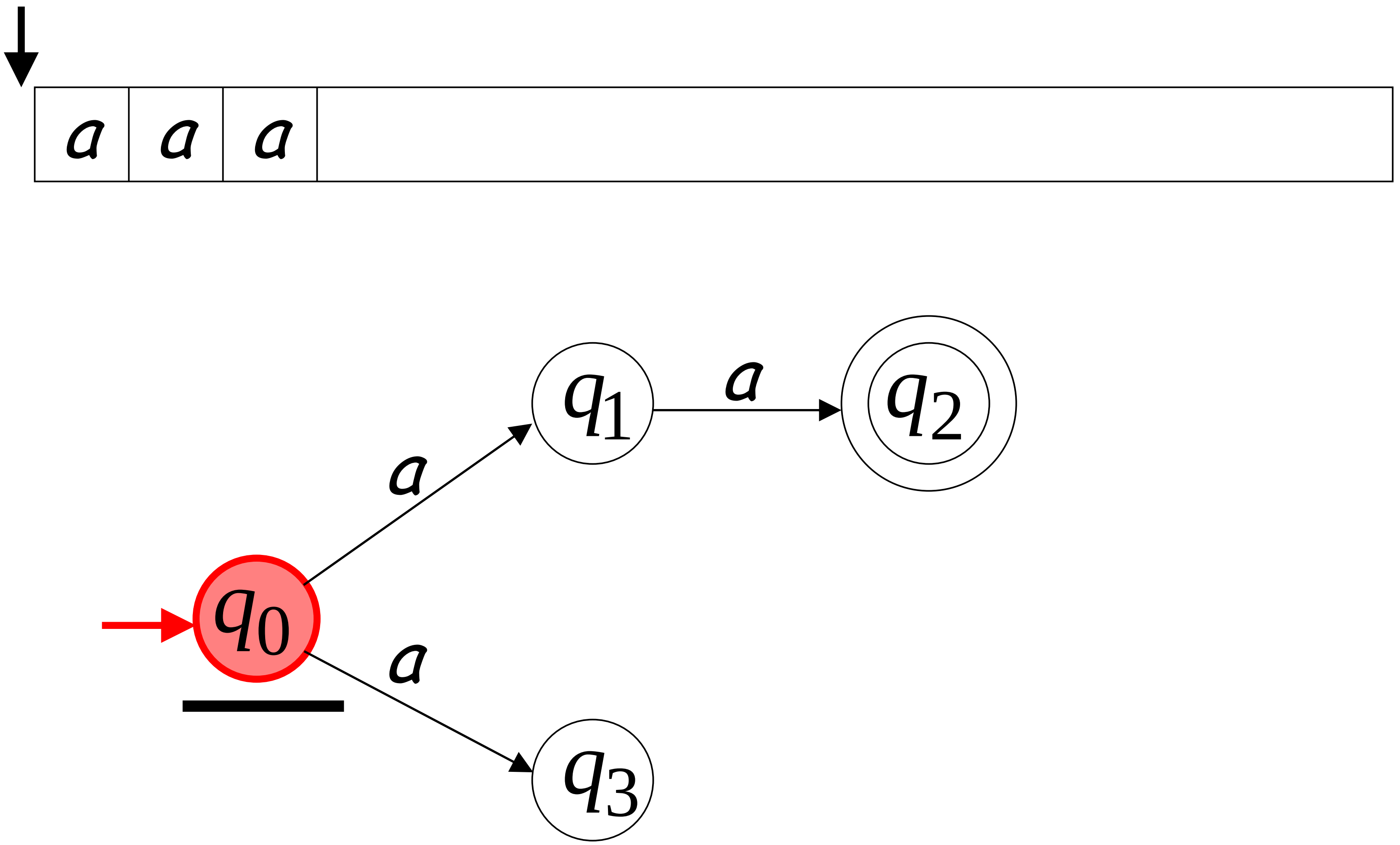


Input cannot be consumed

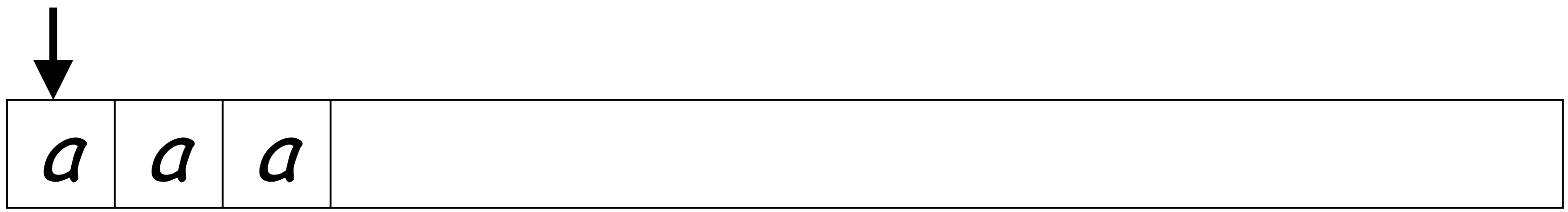
"reject"

Automaton halts

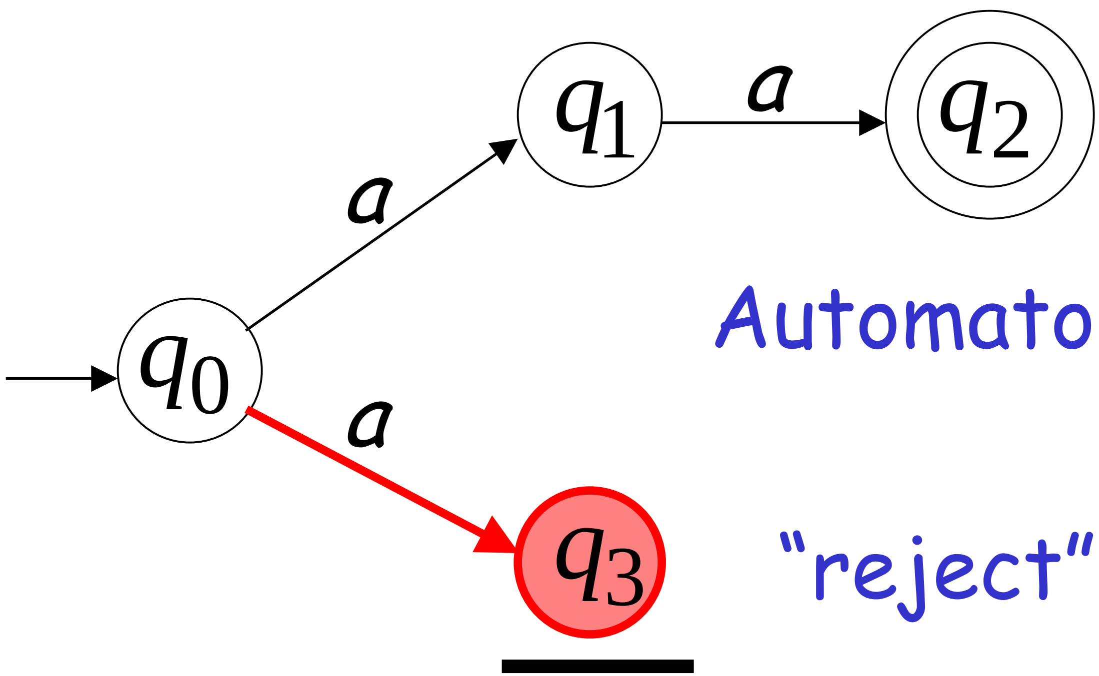
Second Choice



Second Choice



Input cannot be consumed



Automaton halts

"reject"

An NFA rejects a string:
if there is no computation of the NFA that accepts the string.

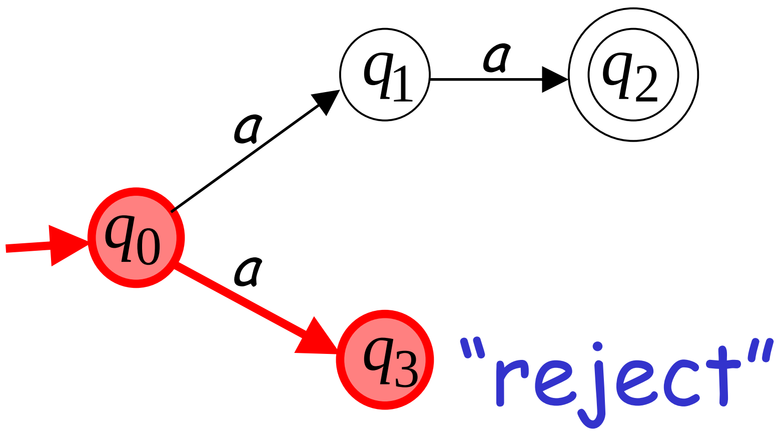
For each computation:

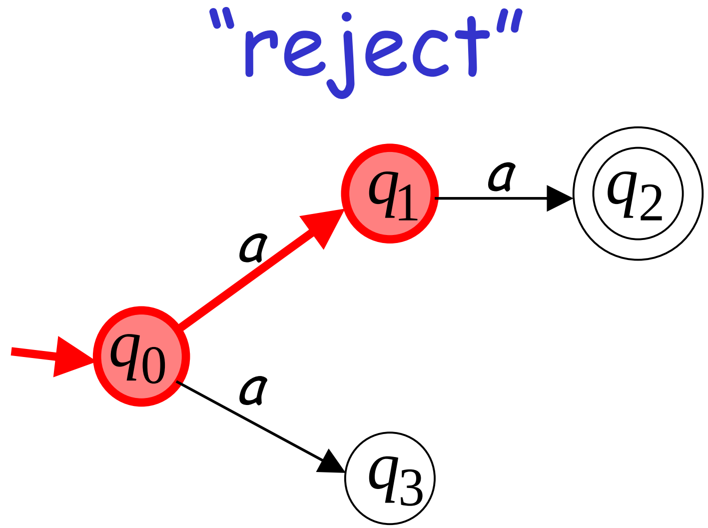
- All the input is consumed and the automaton is in a non final state

OR

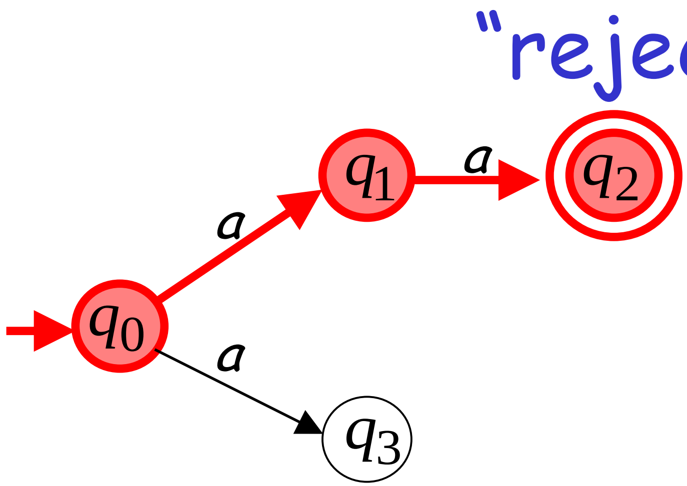
- The input cannot be consumed

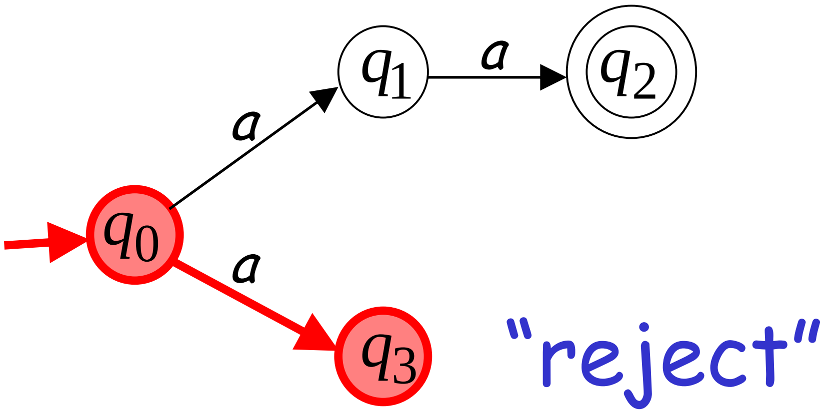
a is rejected by the NFA:





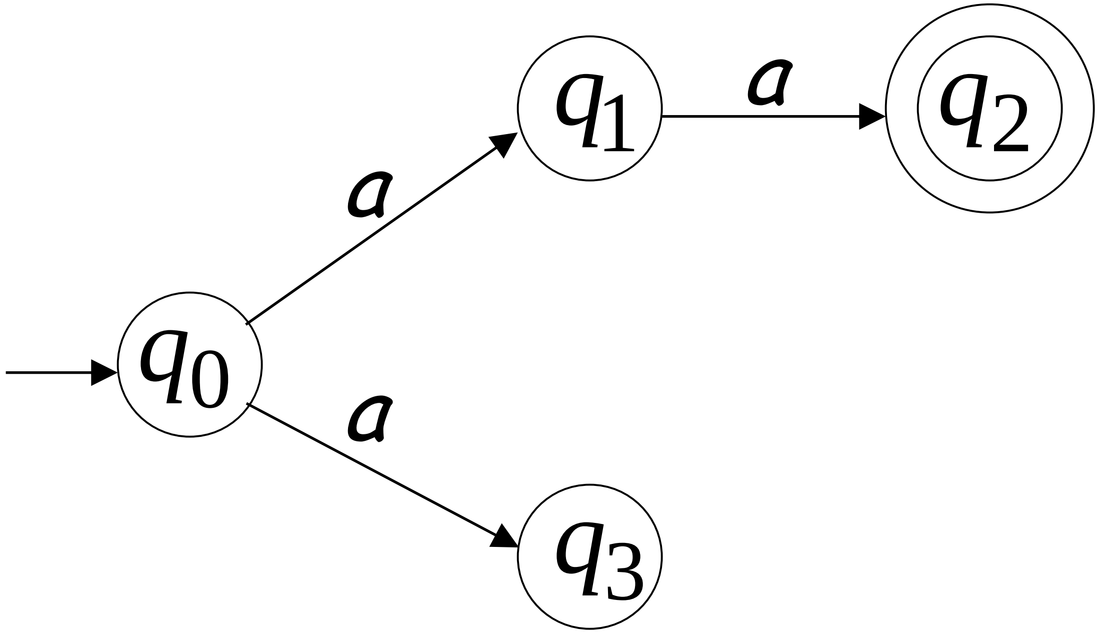
All possible computations lead to rejection aaa is rejected by the NFA:



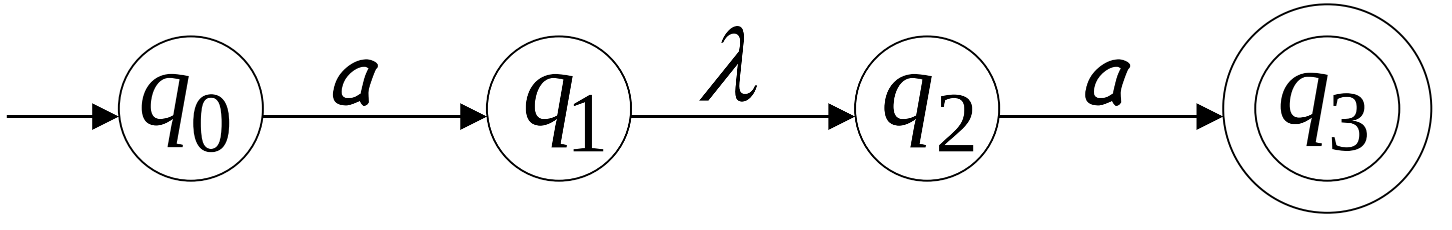


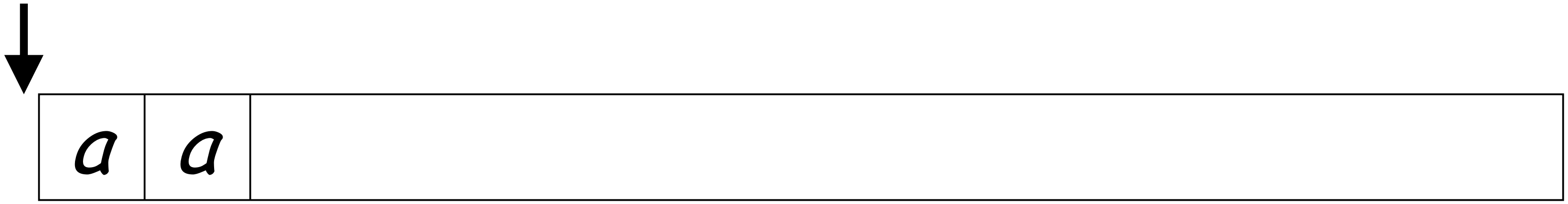
All possible computations lead to rejection

Language accepted: $L = \{aa\}$

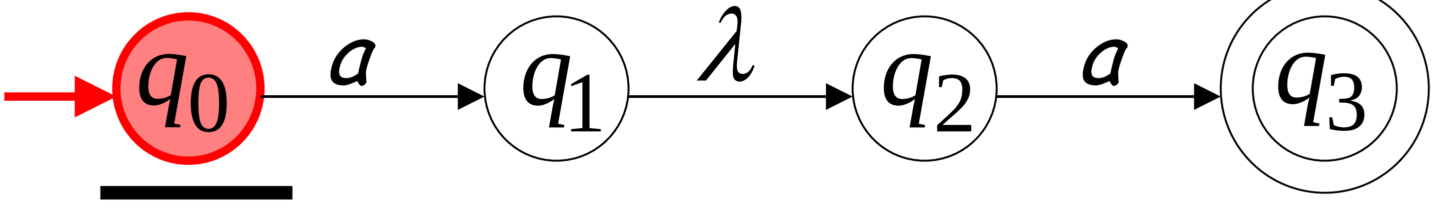


Lambda Transitions

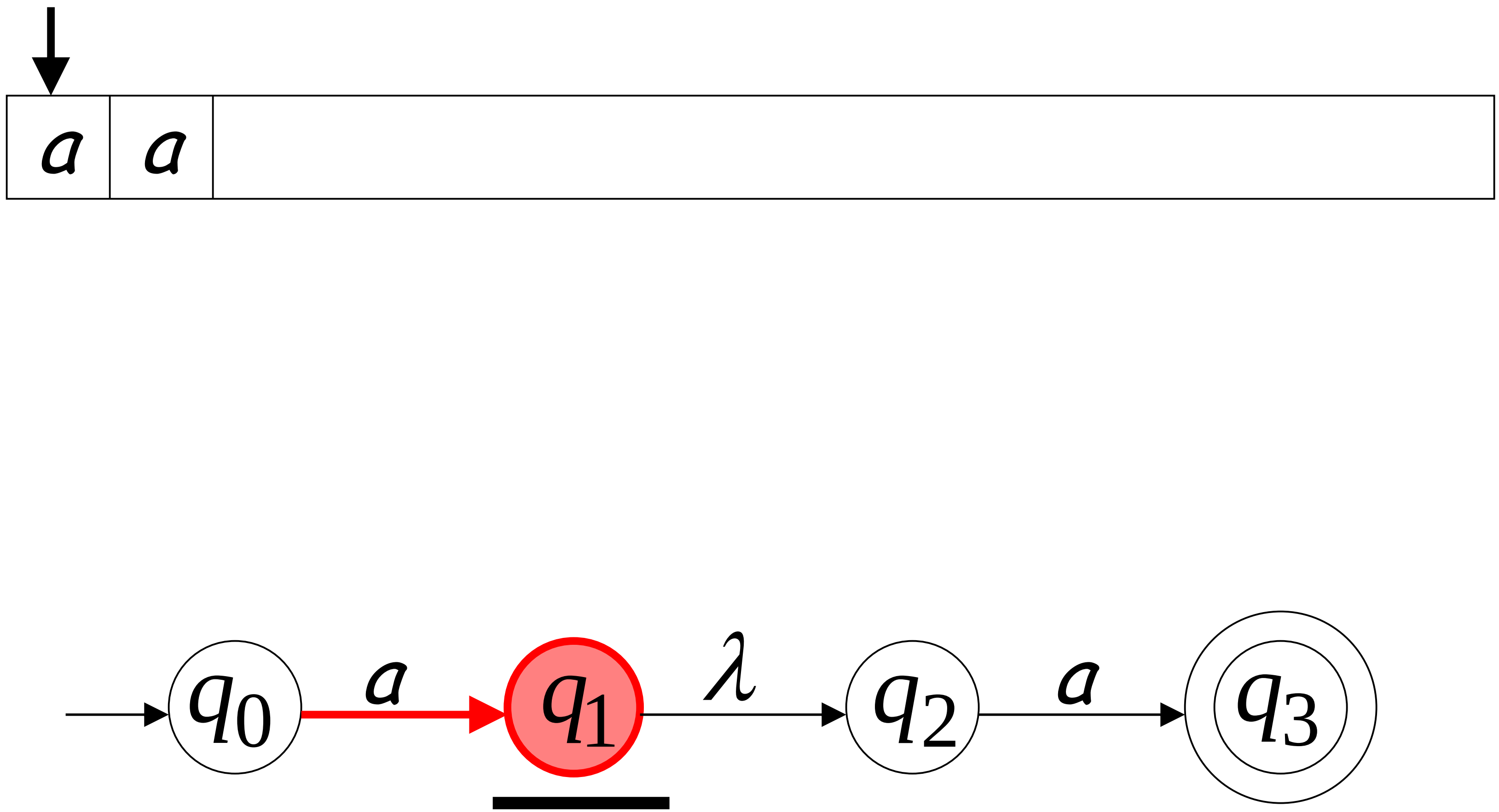




a
a

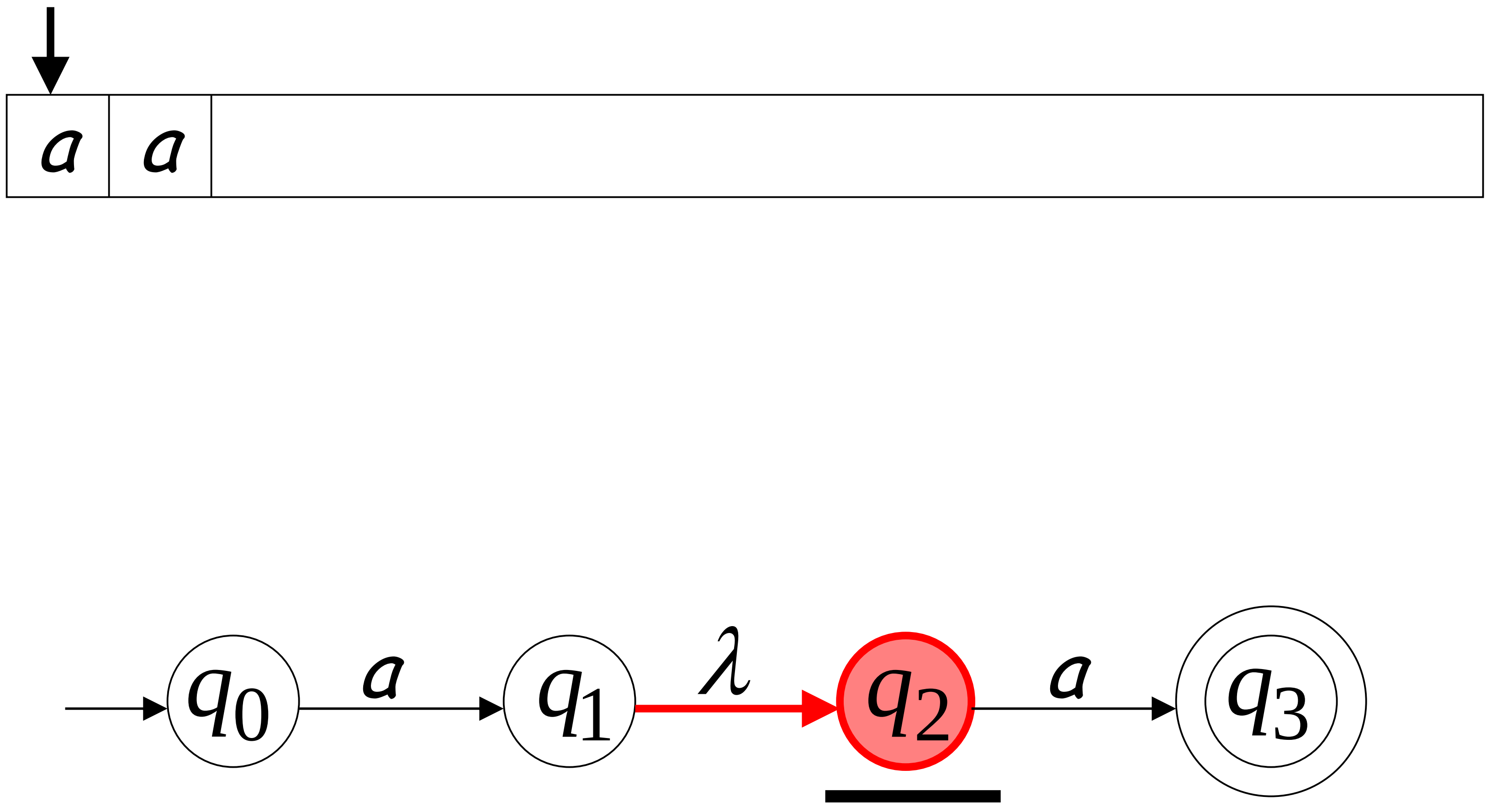


q_0
a
q_1
λ
q_2
a
q_3

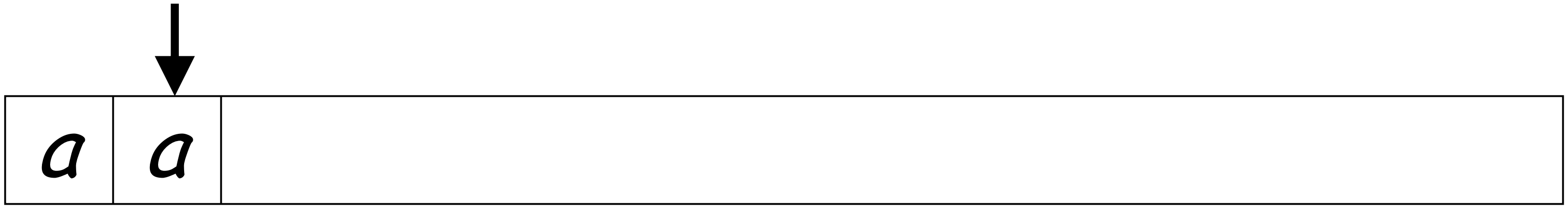


a
a
a
λ
a
q_0
q_1
q_2
q_3

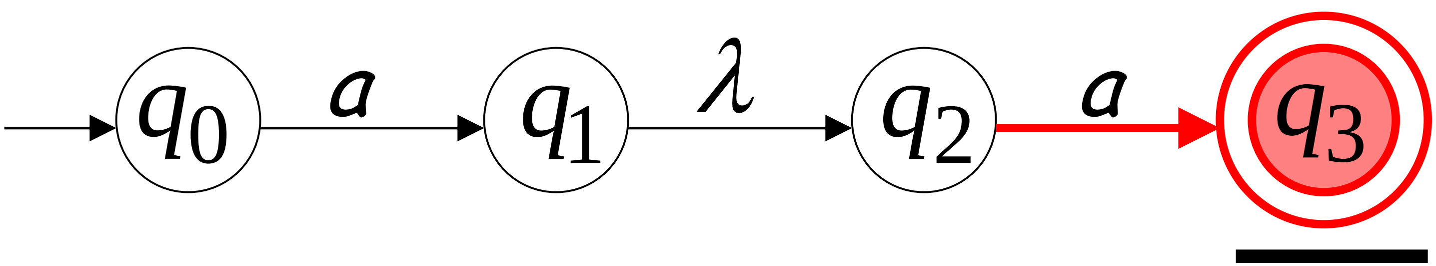
input tape head does not move



all input is consumed

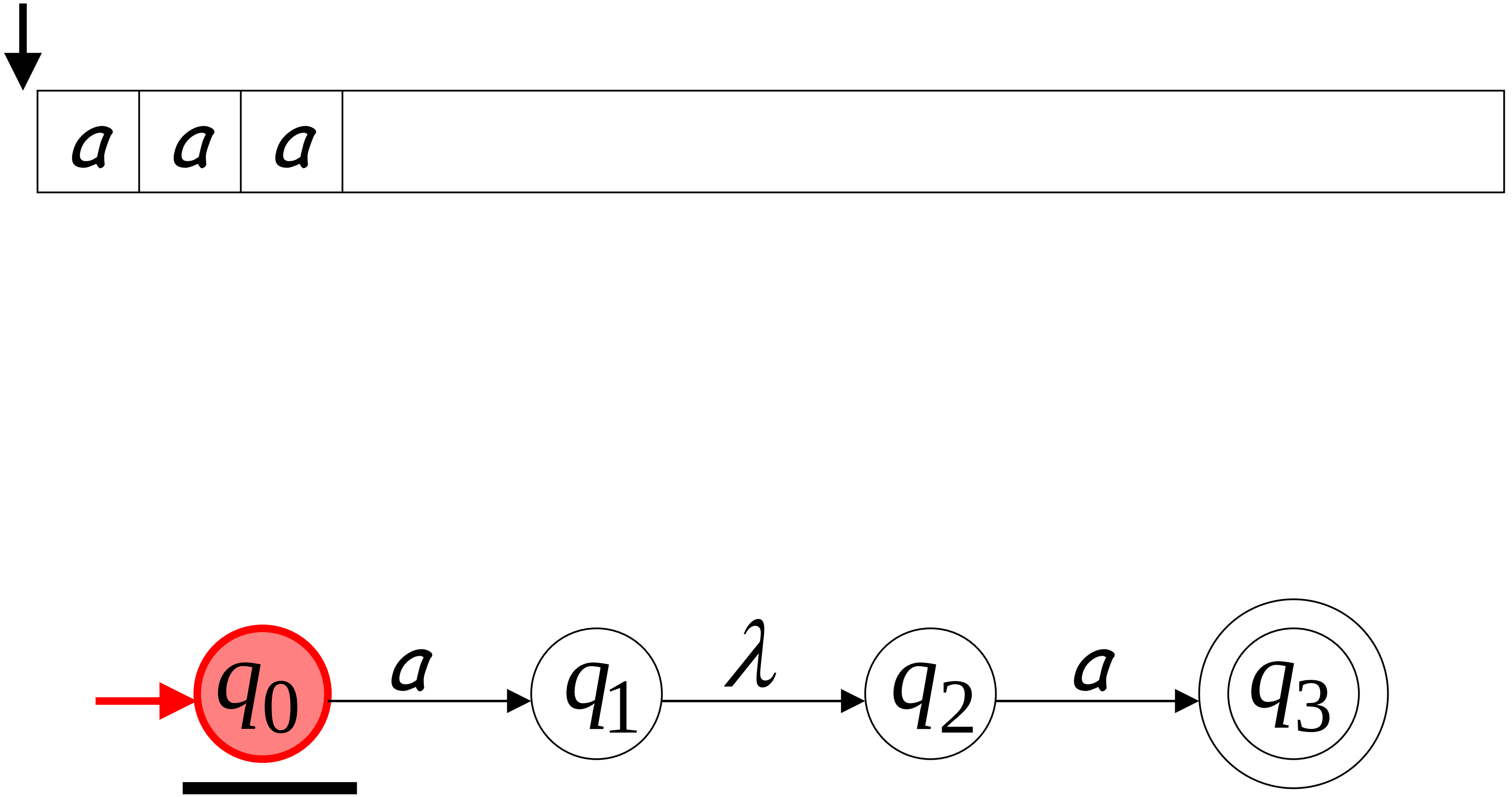


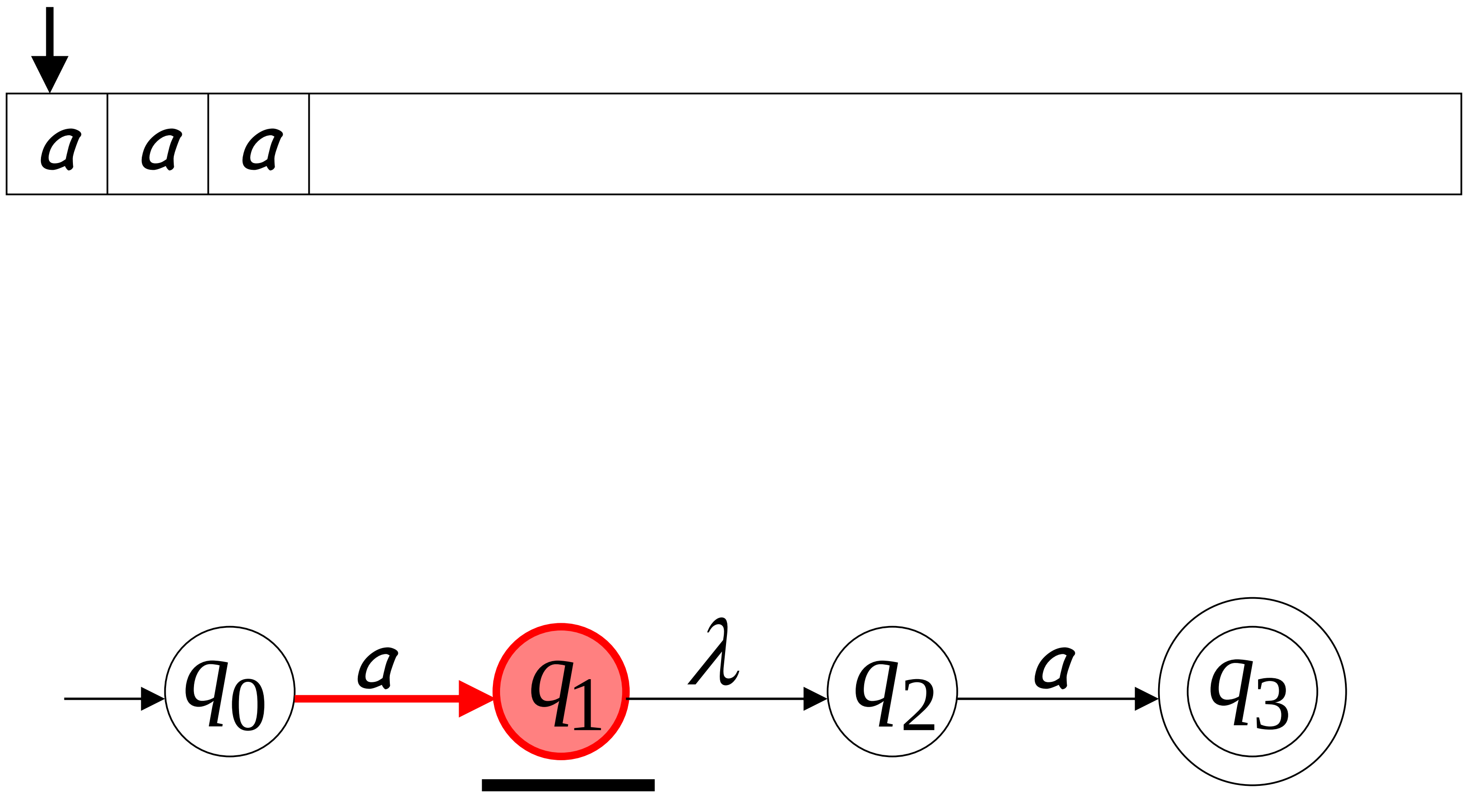
"accept"



String aa is accepted

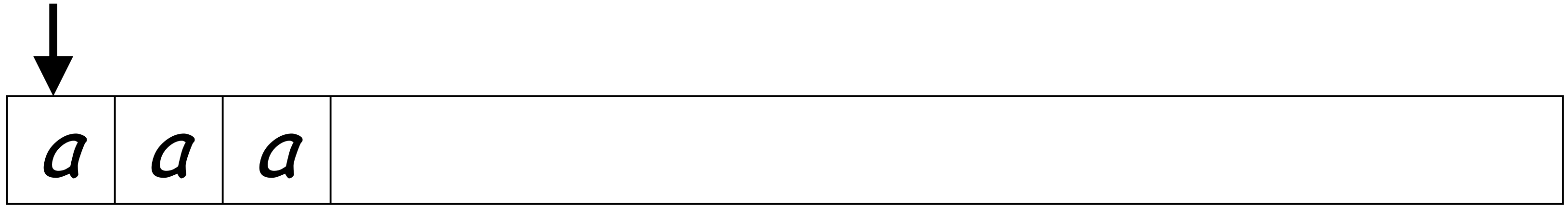
Rejection Example

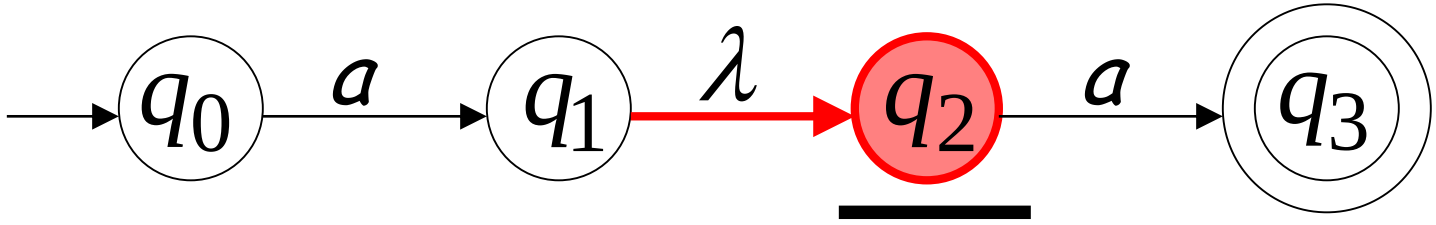




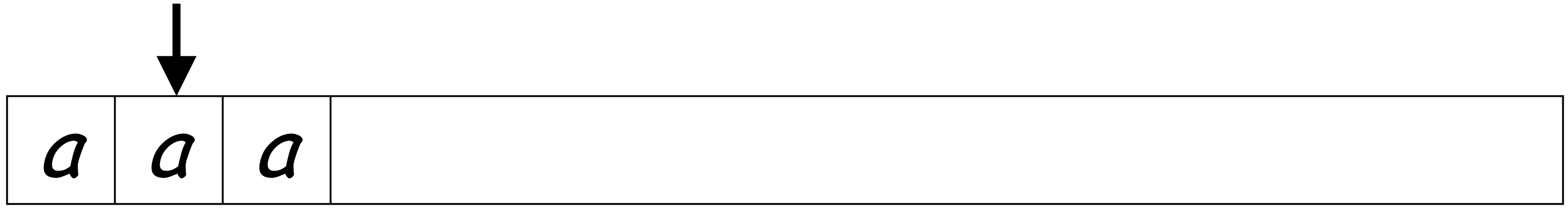
a a a
q0 a q1 λ q2 a q3

(read head doesn't move)

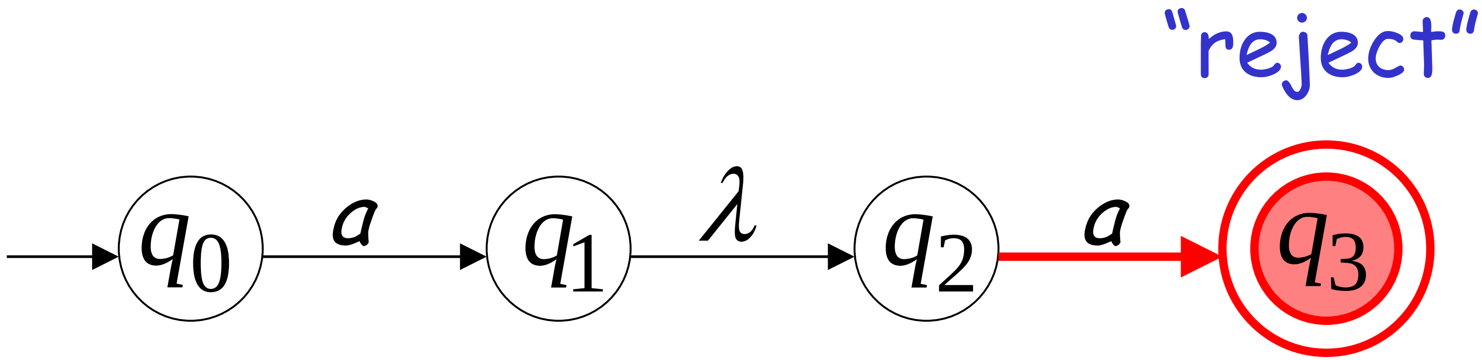




Input cannot be consumed

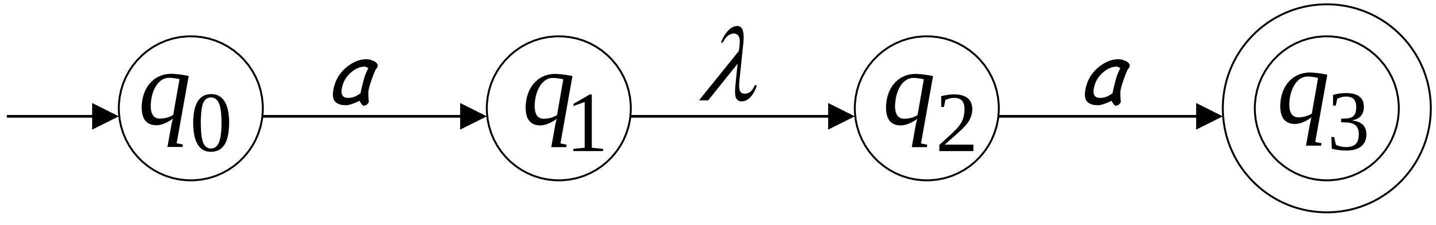


Automaton halts

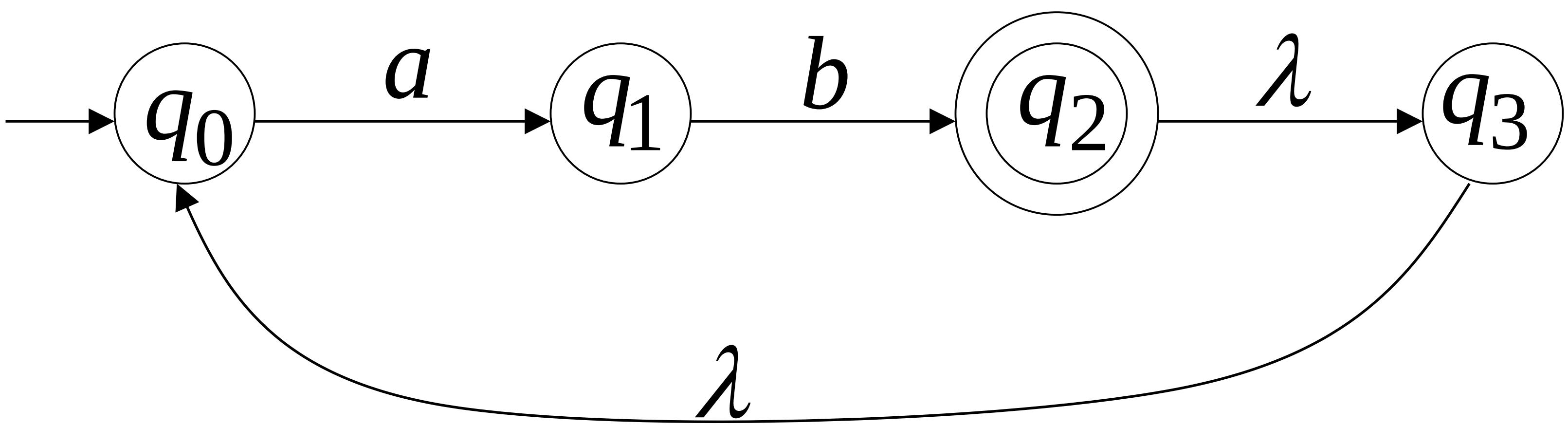


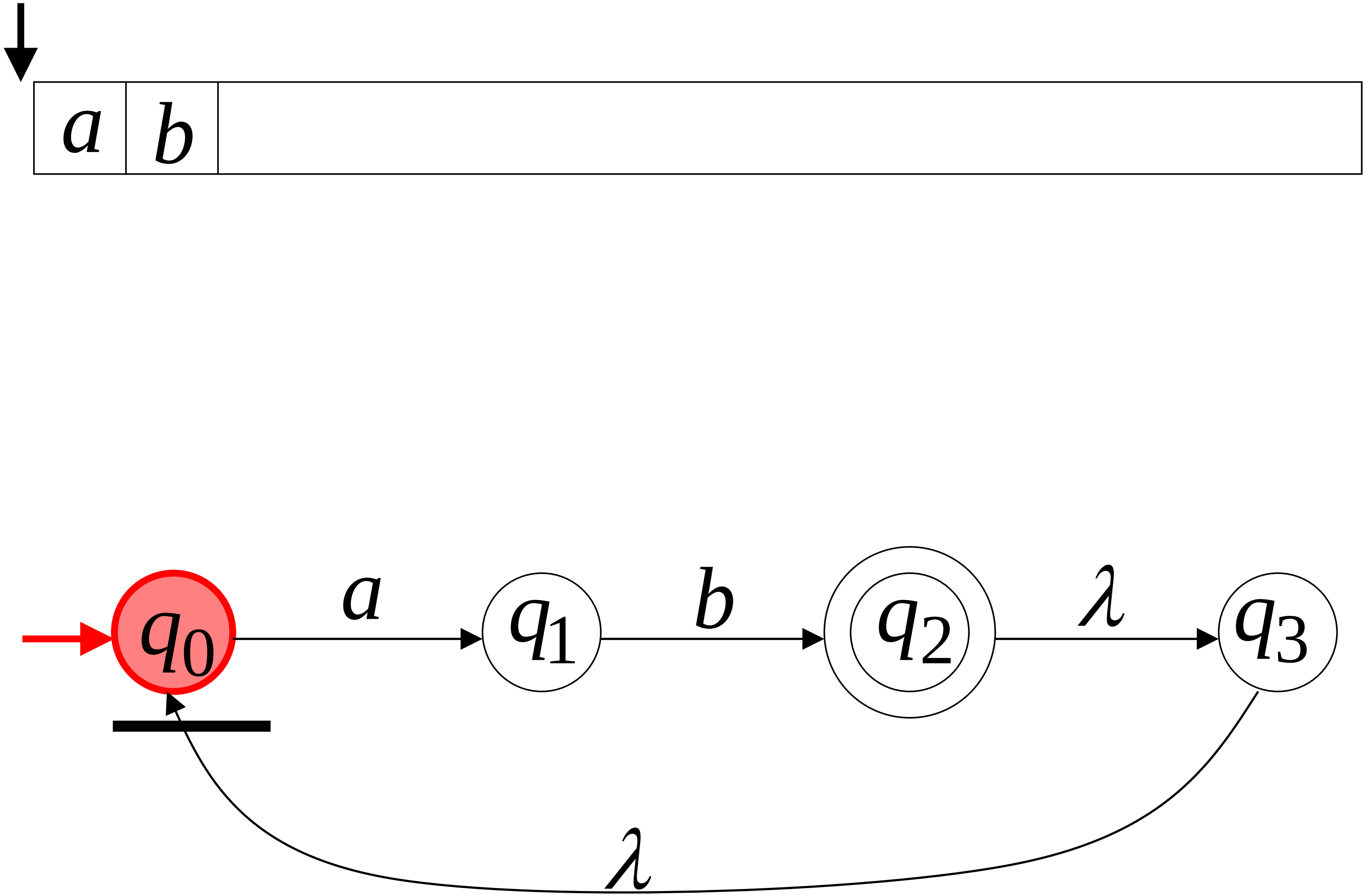
String aaa is rejected

Language accepted: $L=\{aa\}$

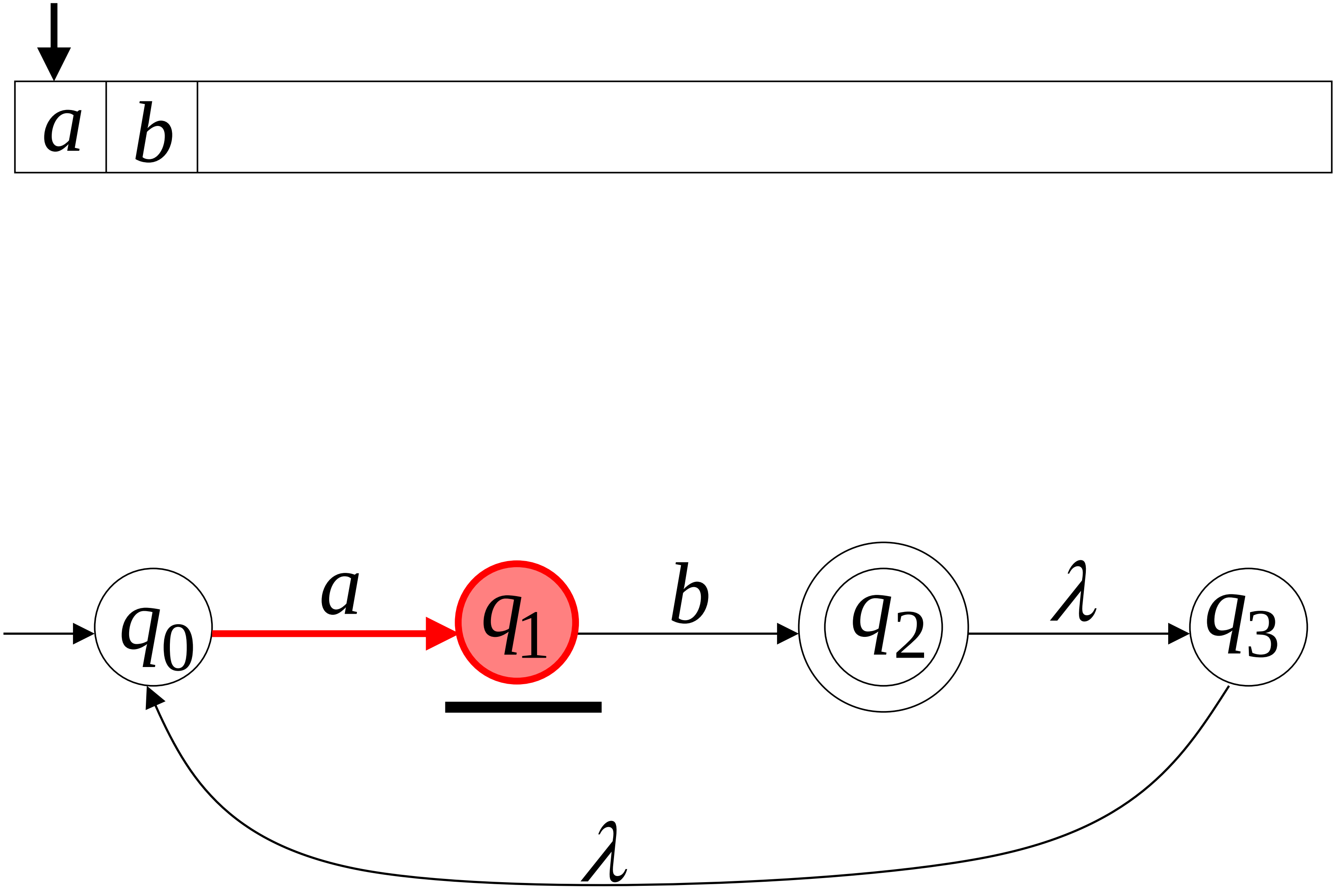


Another NFA Example

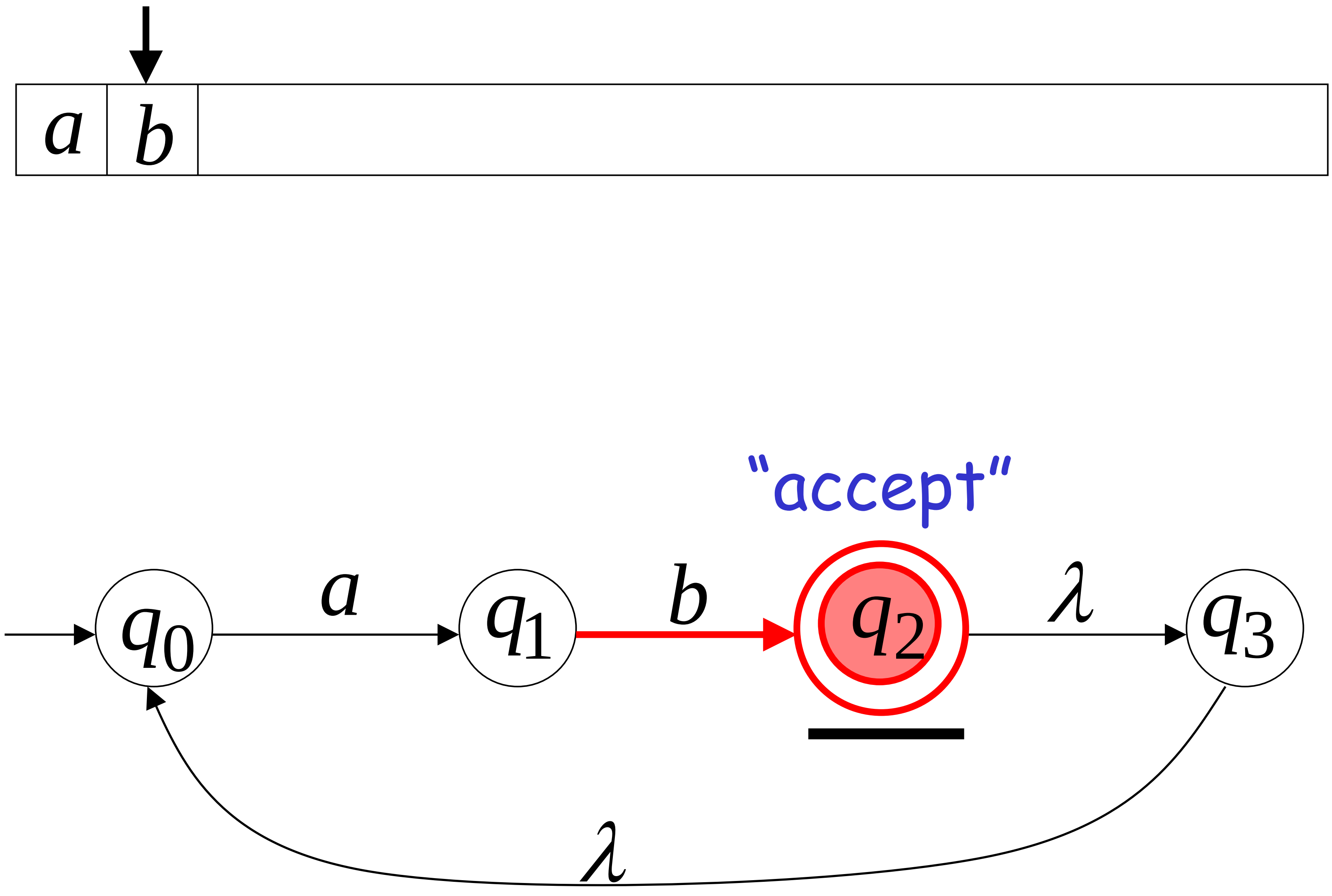




a
b
q0
a
q1
b
q2
λ
q3
λ

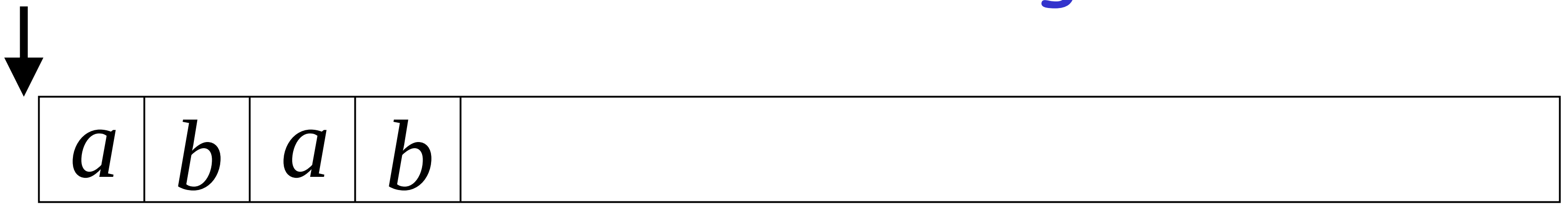


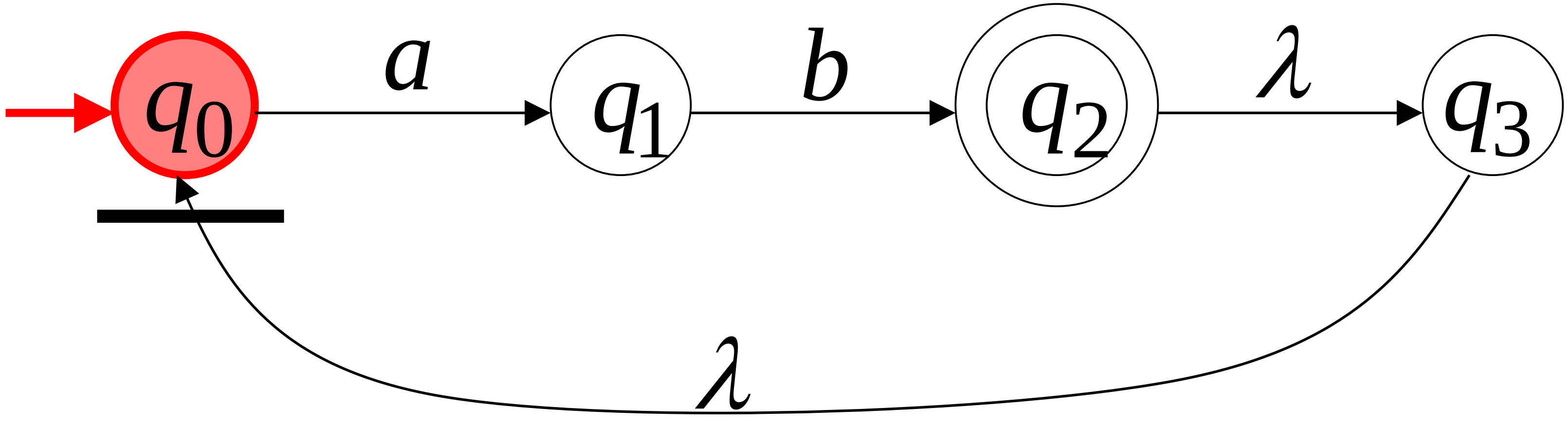
a
b
a
b
λ
λ
q0
q1
q2
q3

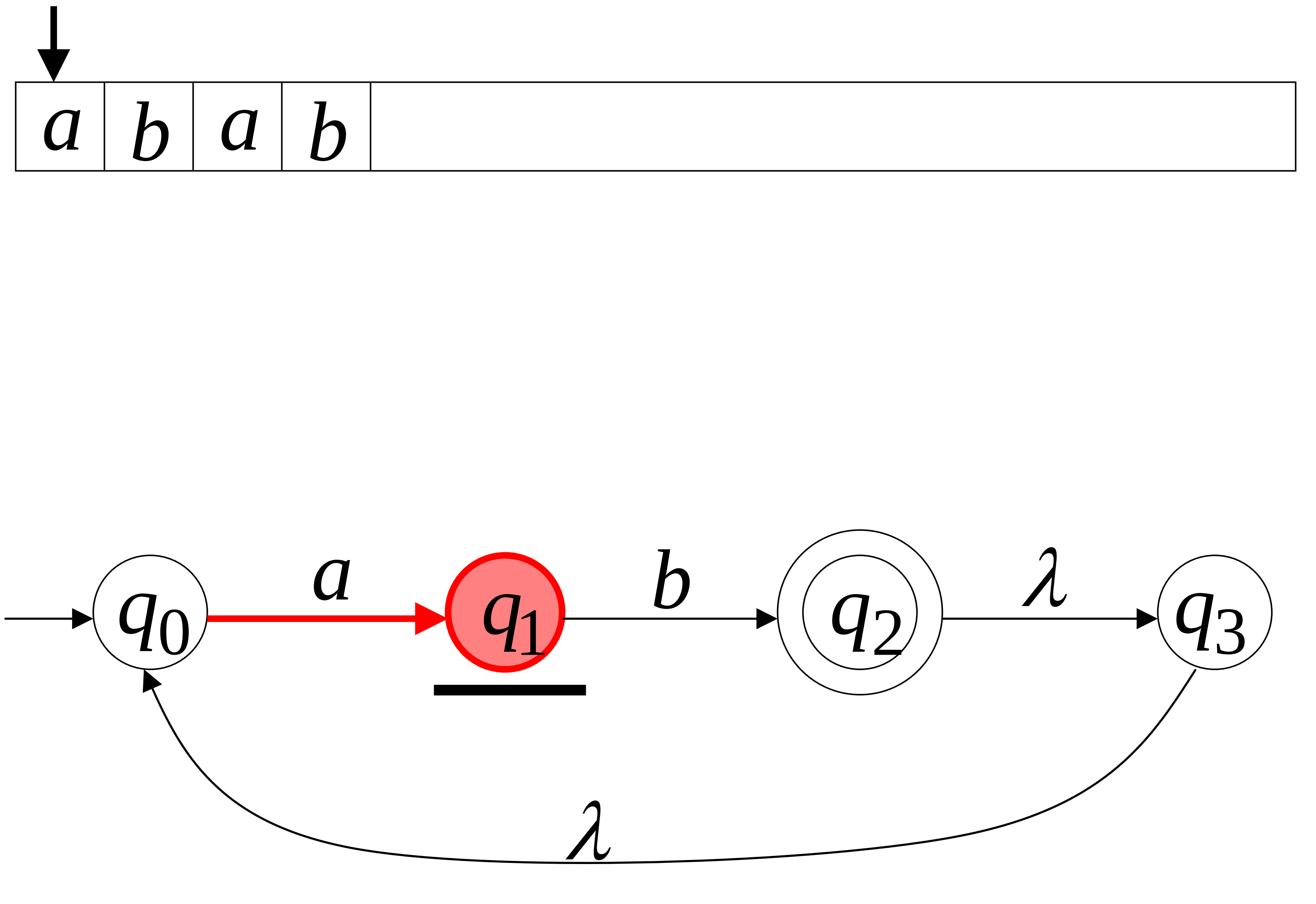


a
b
"accept"
q0
a
q1
b
q2
λ
q3
λ

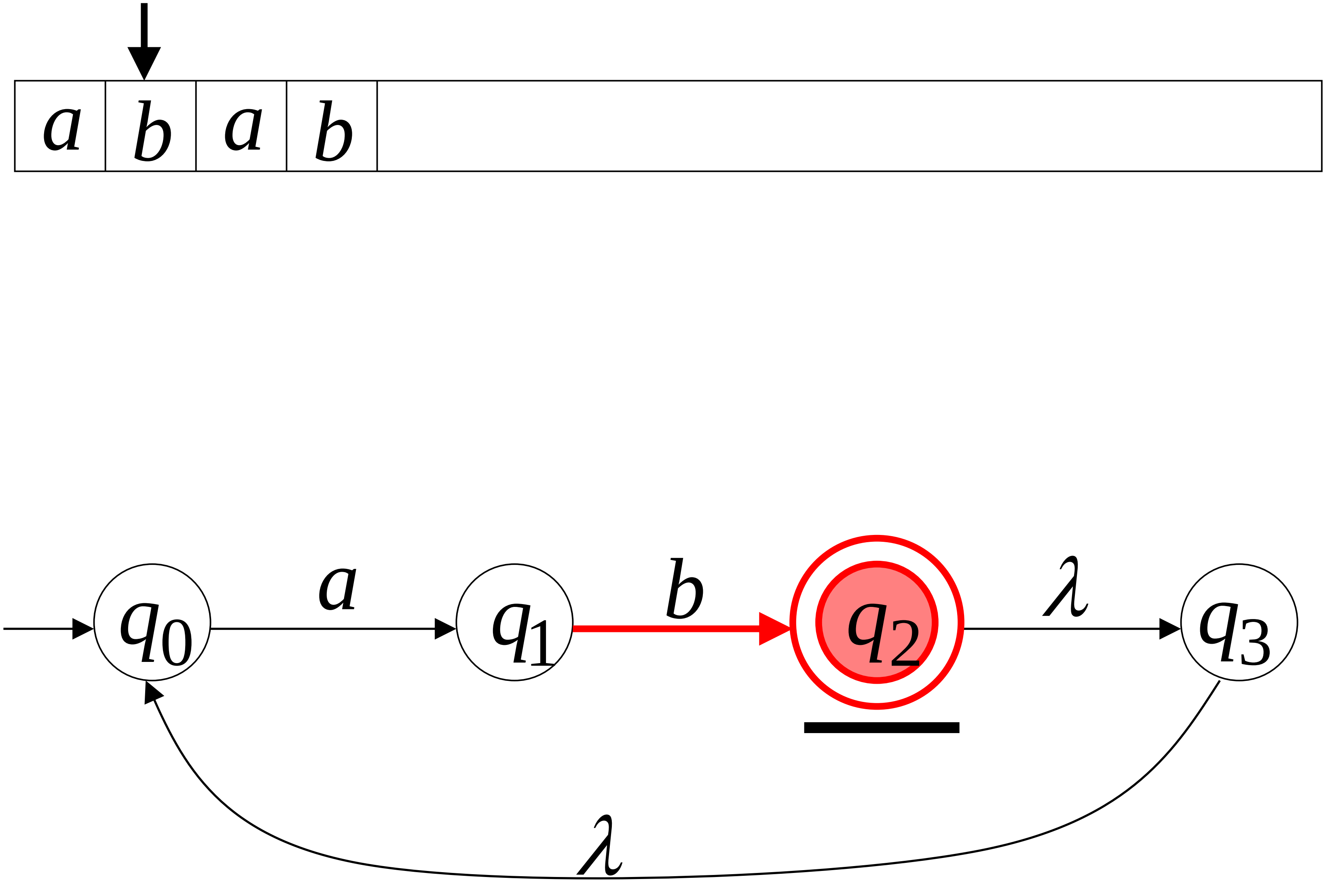
Another String



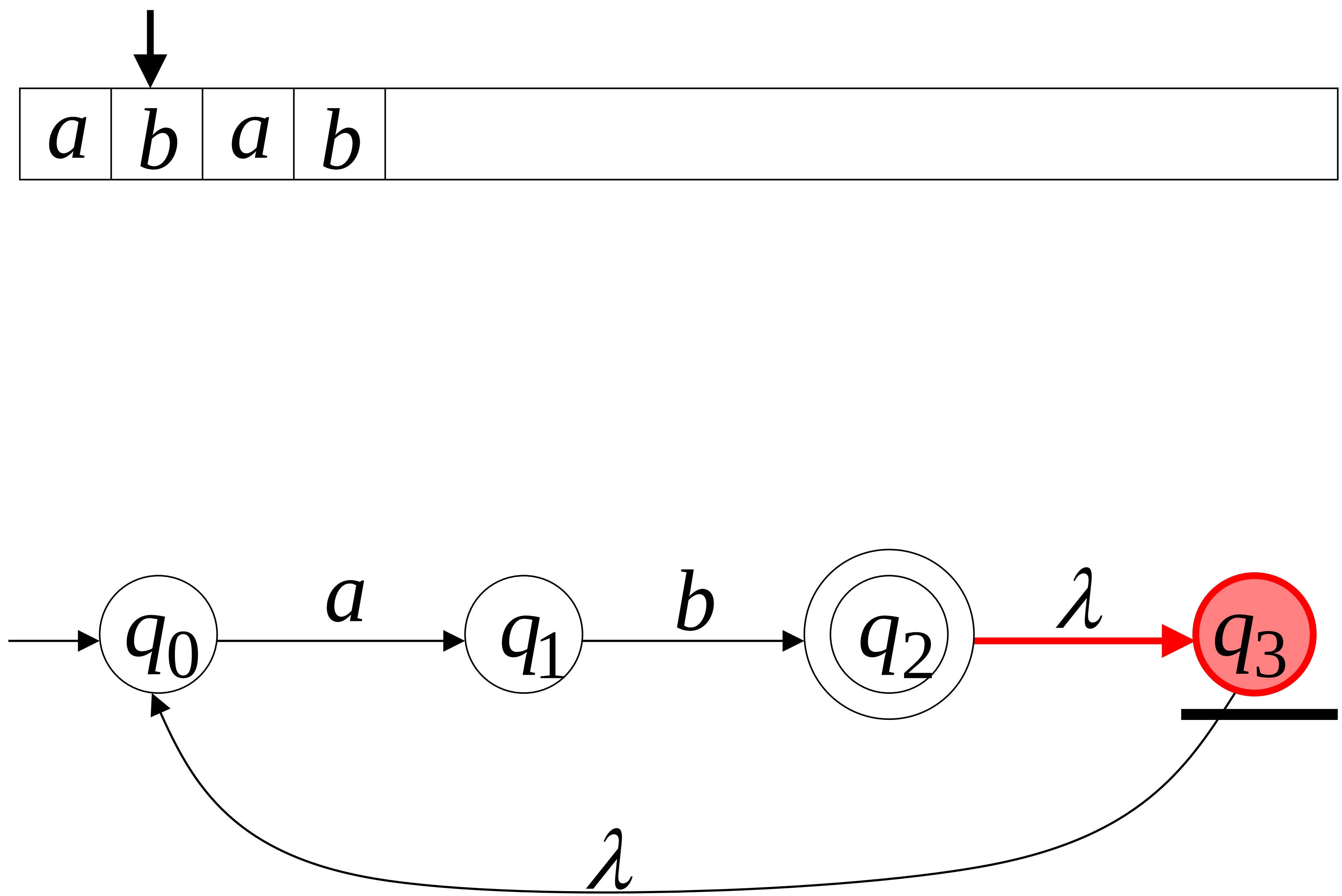




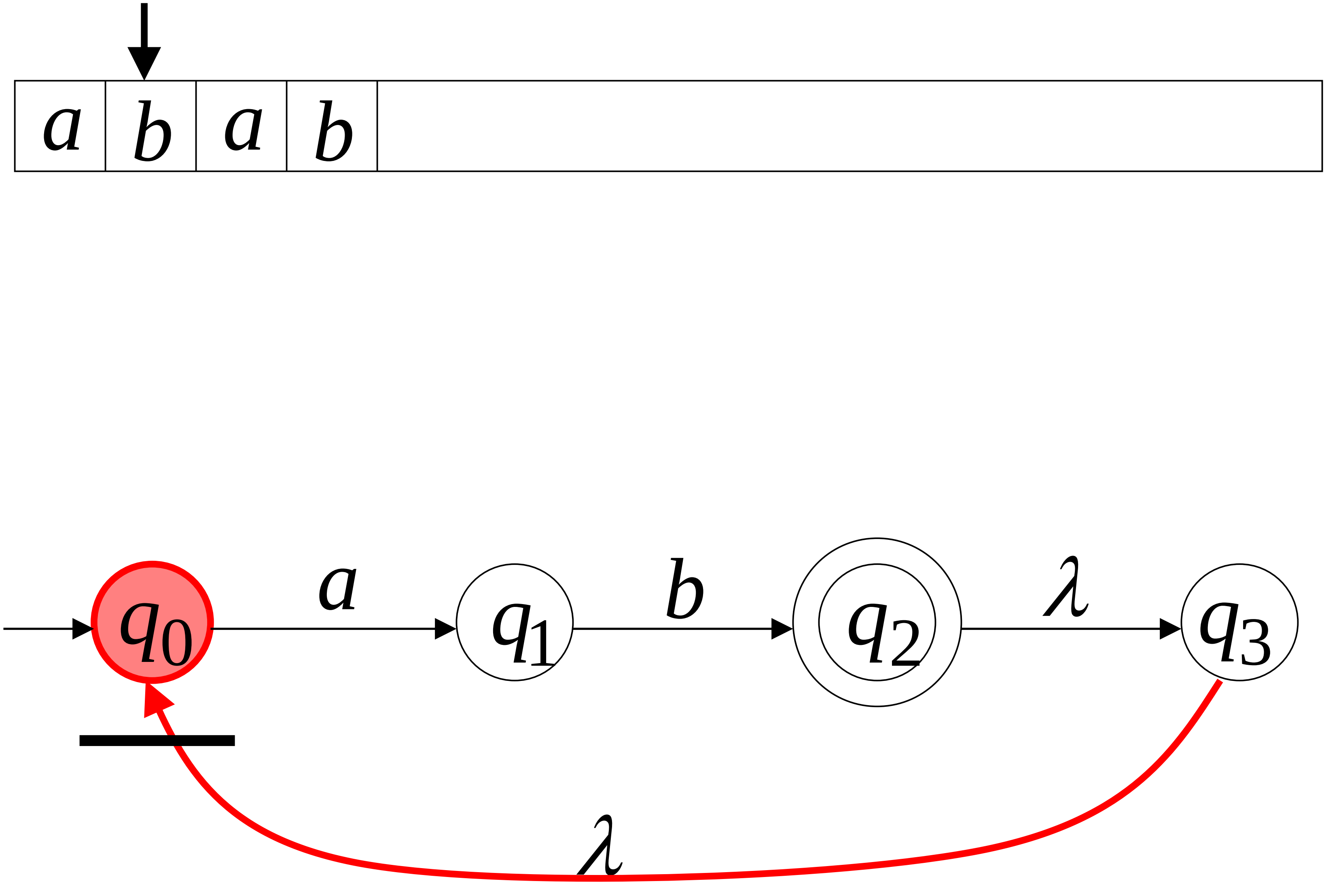
a b a b
a
b
λ
λ
q0
q1
q2
q3

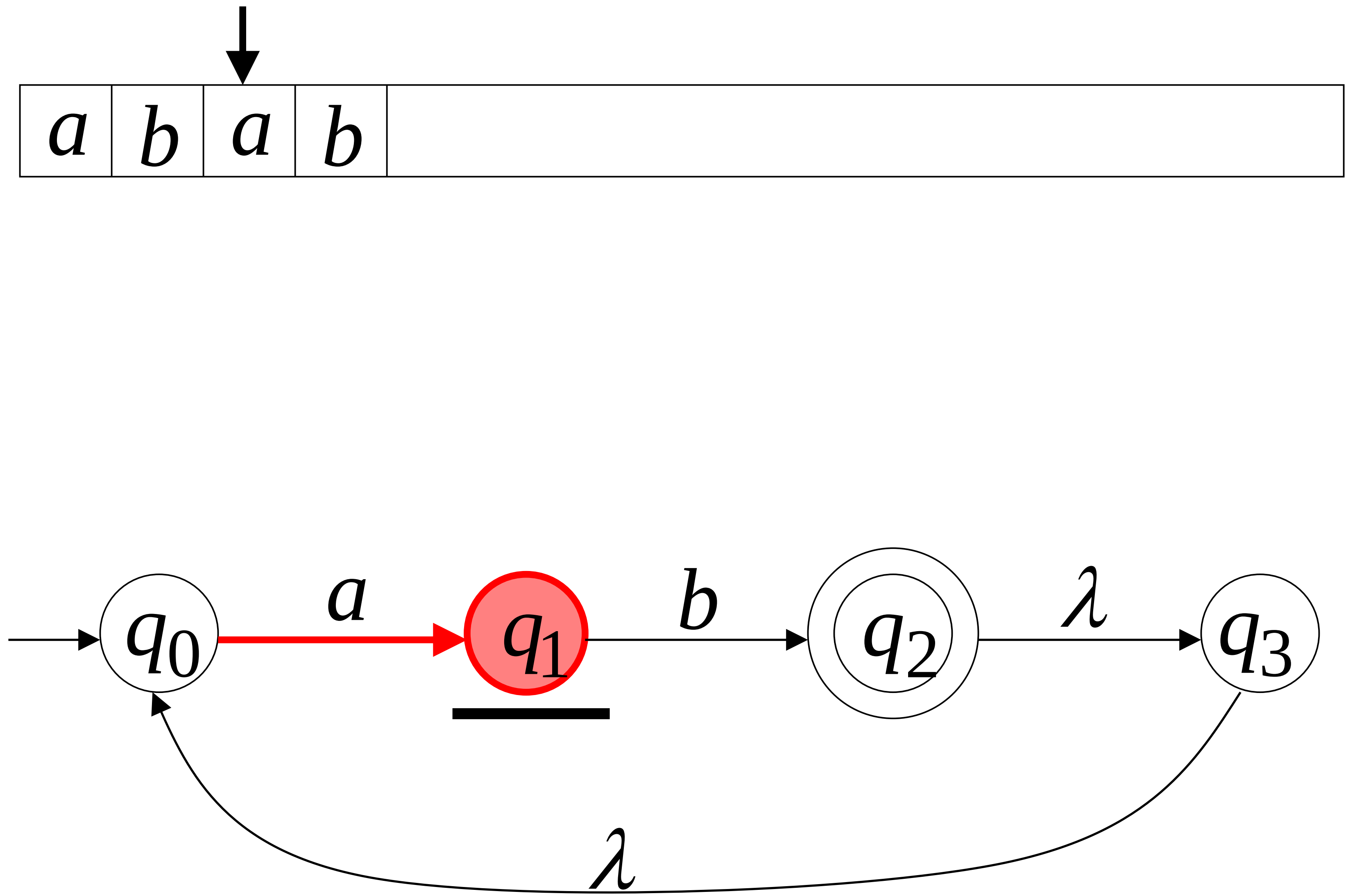


a
b
a
b
a
b
λ
λ
q_0
q_1
q_2
q_3

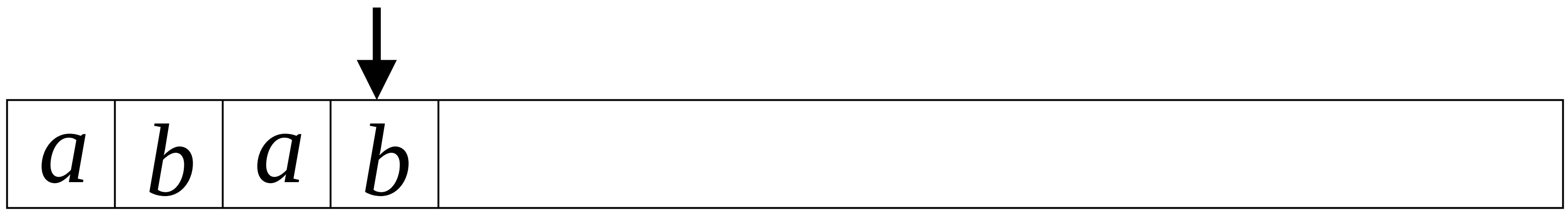


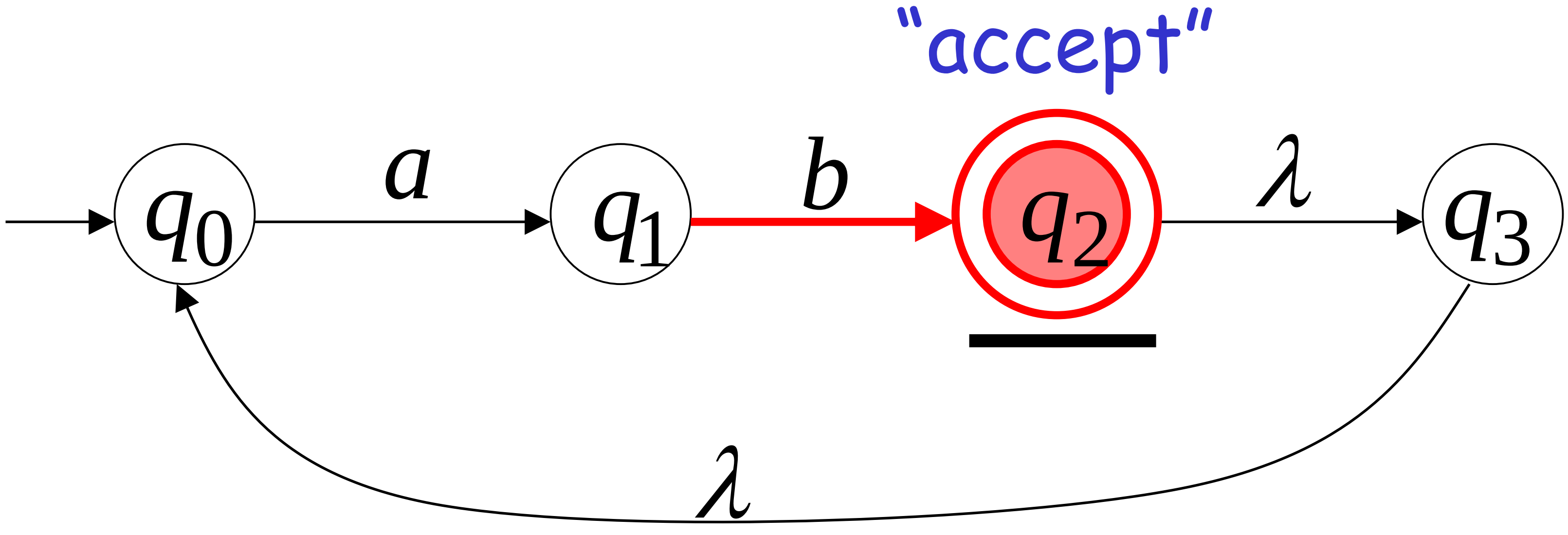
a b a b
a
b
λ
q0 q1 q2 q3
λ


a
b
a
b
q_0
a
q_1
b
q_2
λ
q_3
λ



a
b
a
b
a
b
λ
λ

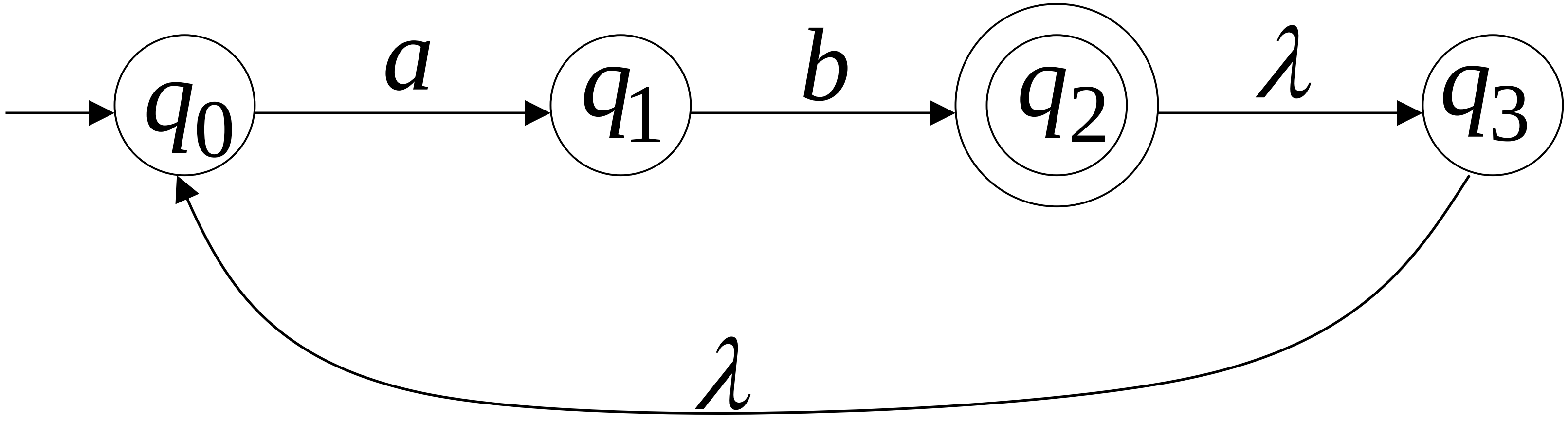

a
b
a
b


"accept"
q_0
a
q_1
b
q_2
λ
q_3
λ

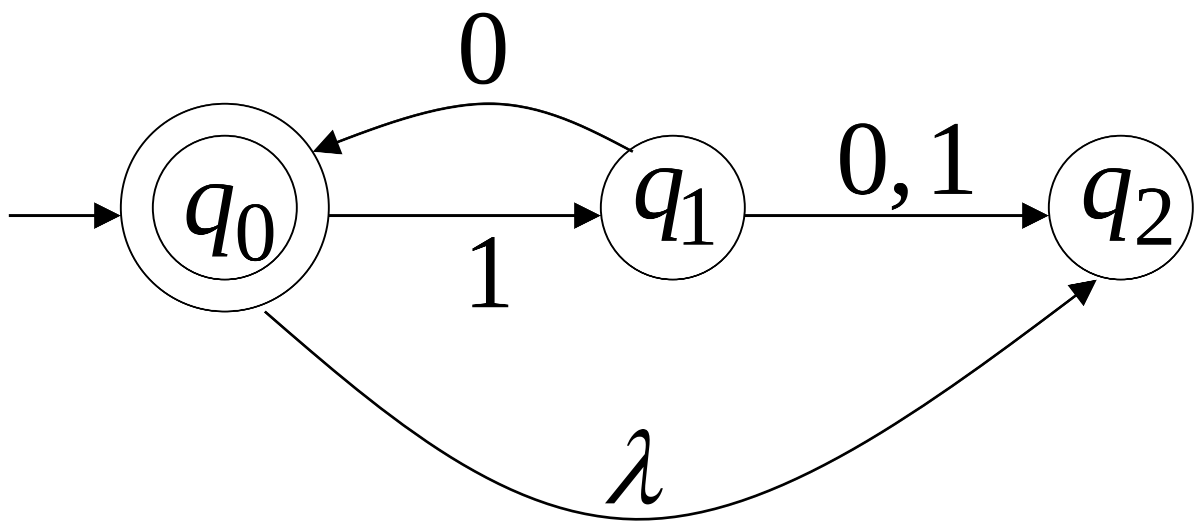
Language accepted

$$L = \{ab,\ abab,\ ababab,\ \ldots\}$$

$$= \{ab\}^{+}$$



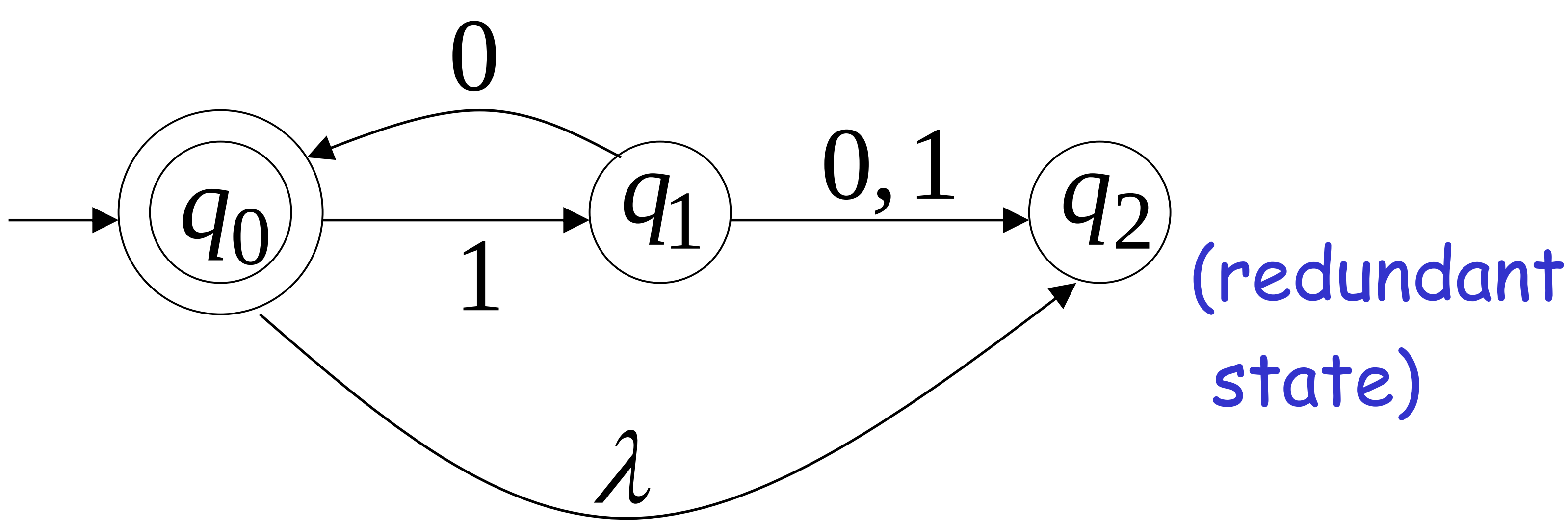
Another NFA Example



Language accepted

$$L(M) = \{\lambda,\ 10,\ 1010,\ 101010,\ \ldots\}$$

$$= \{10\}^*$$

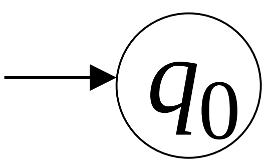


Remarks:

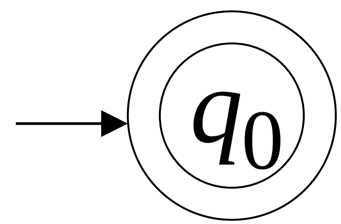
- The λ symbol never appears on the input tape

- Simple automata:

M_1



M_2

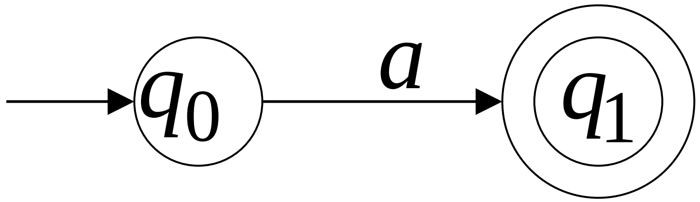


$L(M_1) = \{\}$

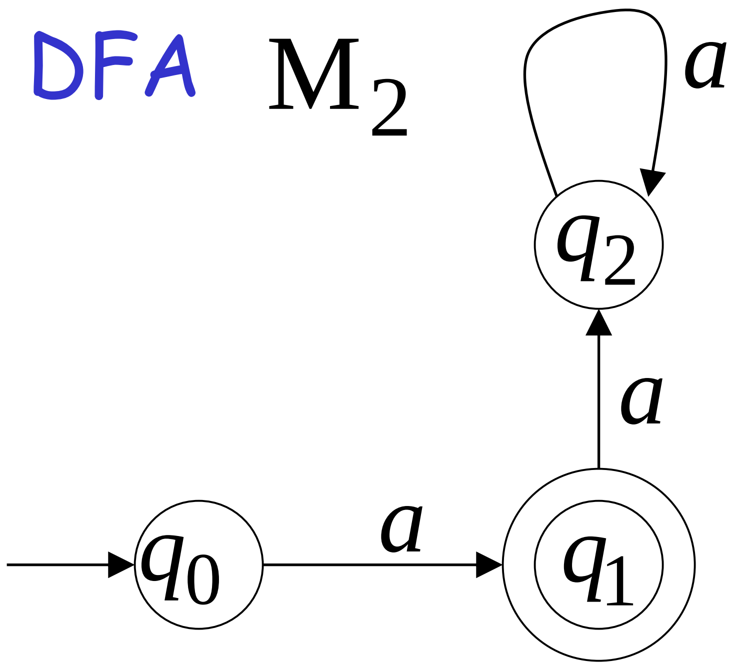
$L(M_2) = \{\lambda\}$

- NFAs are interesting because we can express languages easier than DFAs

NFA M_1



DFA M_2



$$L(M_1) = \{a\}$$

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

Formal Definition of NFAs

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

$Q:$ Set of states, i.e. $\{q_0, q_1, q_2\}$

$\Sigma:$ Input aplhabet, i.e. $\{a, b\}$ $\qquad \lambda \notin \Sigma$

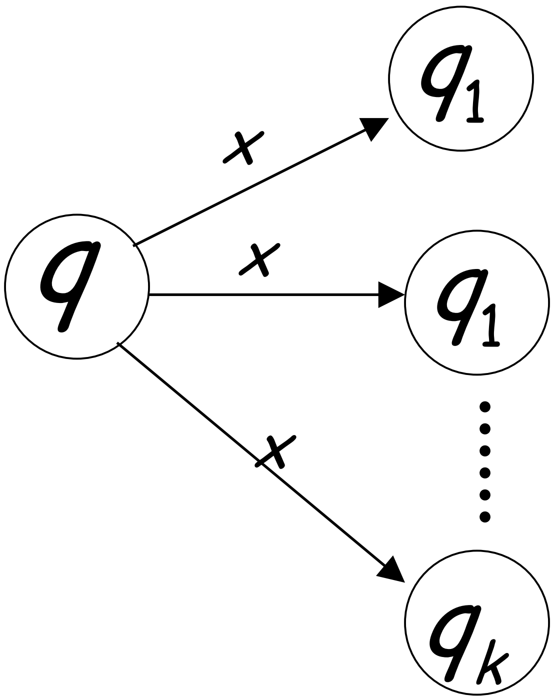
$\delta:$ Transition function

$q_0:$ Initial state

$F:$ Accepting states

Transition Function δ

$$\delta(q, x) = \{q_1, q_2, \ldots, q_k\}$$



resulting states with following one transition with symbol x

Corollary

- *A language is regular if and only if some nondeterministic finite automaton recognizes it*
- *A language is regular if and only if some deterministic finite automaton recognizes it*

Epsilon Transitions

- Extension to NFA – a "feature" called epsilon transitions, denoted by ε, the empty string
- The ε transition lets us spontaneously take a transition, without receiving an input symbol
- Another mechanism that allows our NFA to be in multiple states at once.
  - Whenever we take an ε edge, we must fork off a new "thread" for the NFA starting in the destination state.
- While sometimes convenient, has no more power than a normal NFA
  - Just as a NFA has no more power than a DFA

Formal Notation - Epsilon Transition

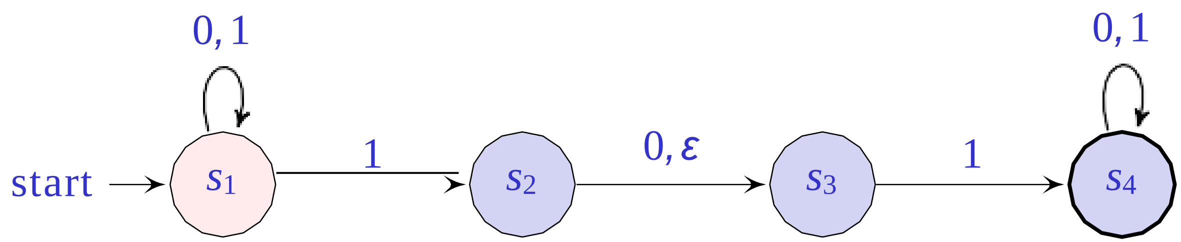
- Transition function δ is now a function that takes as arguments:
  - A state in Q and
  - A member of $\Sigma \cup \{\varepsilon\}$; that is, an input symbol or the symbol ε. We require that ε not be a symbol of the alphabet Σ to avoid any confusion.

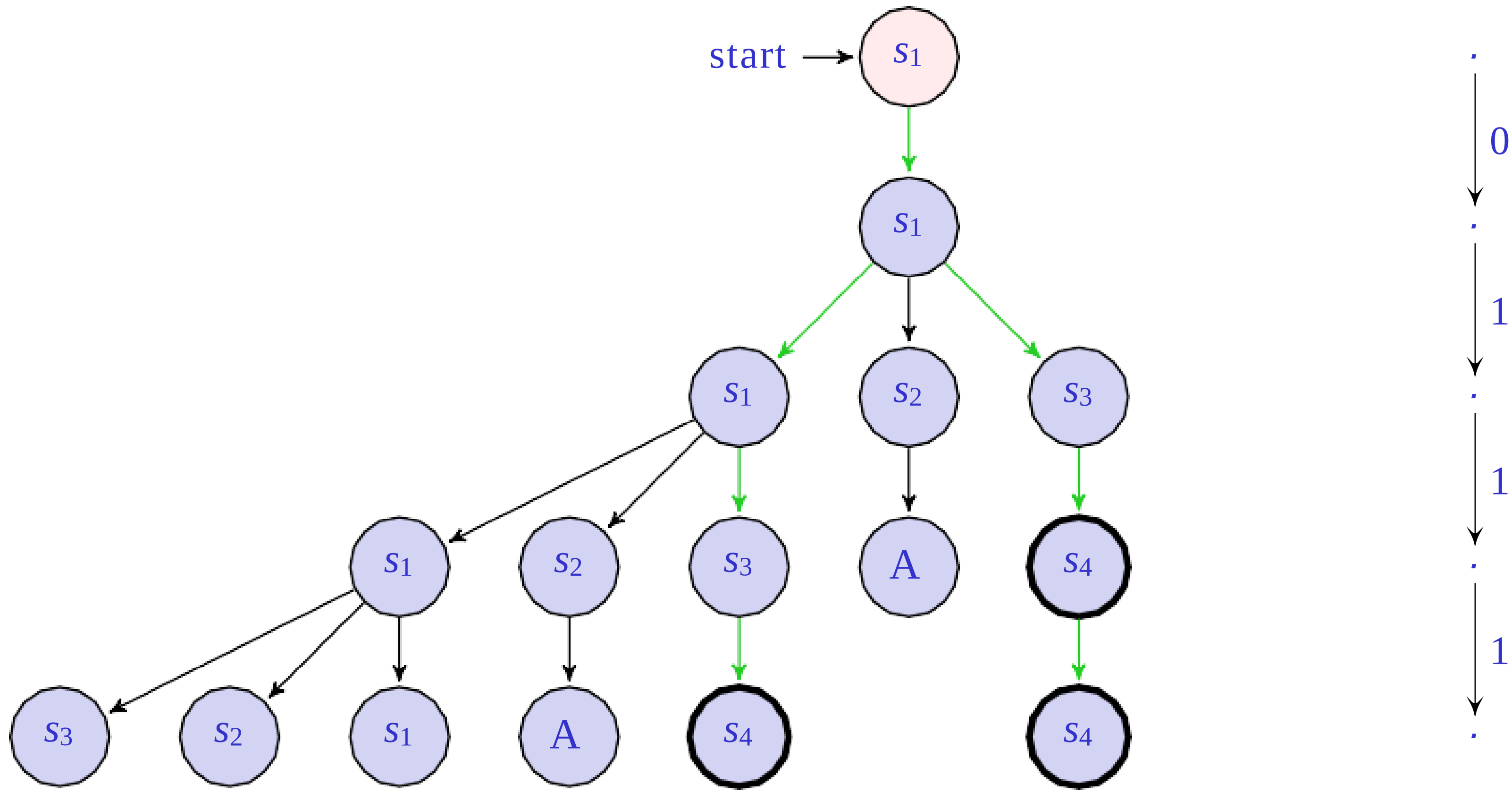
- Design FA Accepting Language of Strings that contain eithe r ab or bba
- Design FA Accepting Language of Strings in which both the numbers of a's and numbers of b's are even.

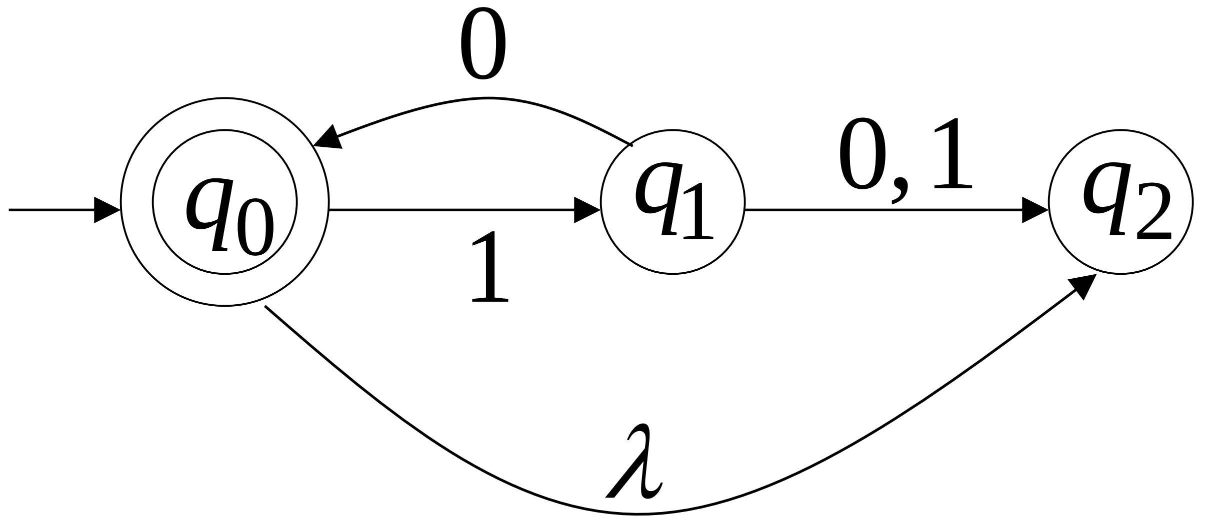
Review

- NFA
- NFA with null transition

Computation of an NFA

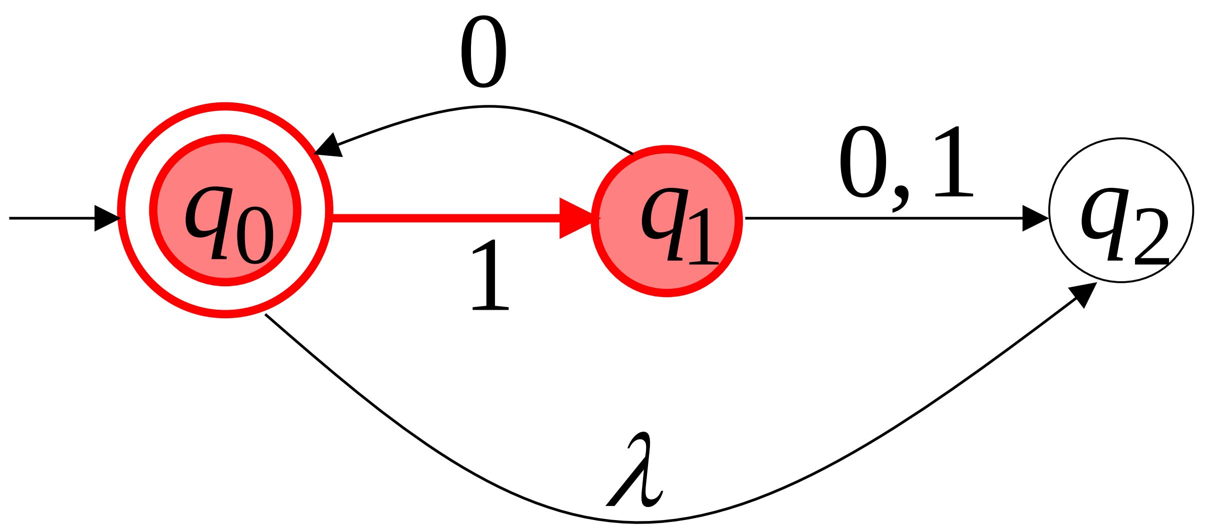




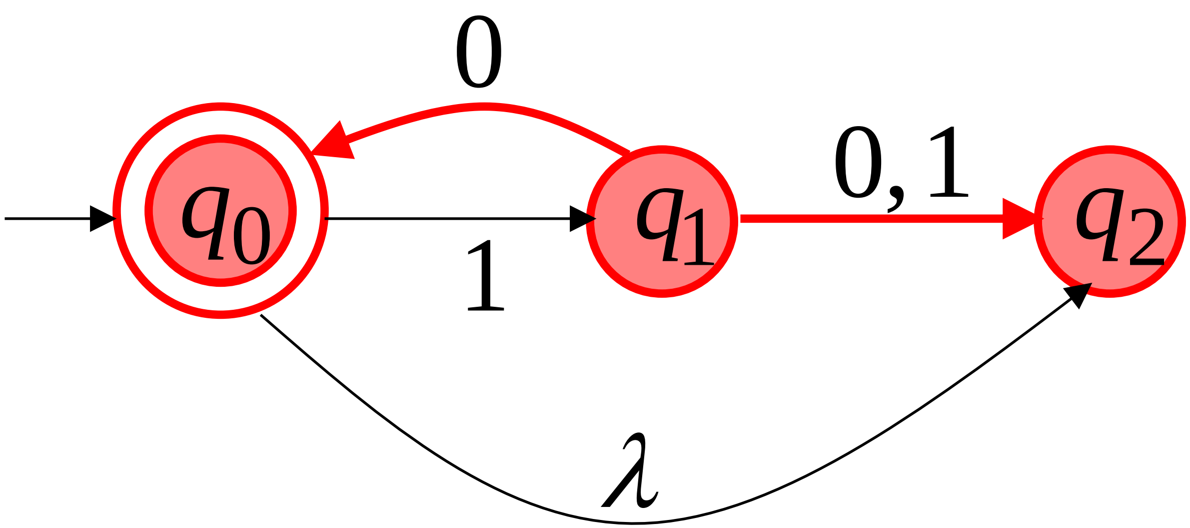


q_0
q_1
q_2
0
1
0, 1
λ

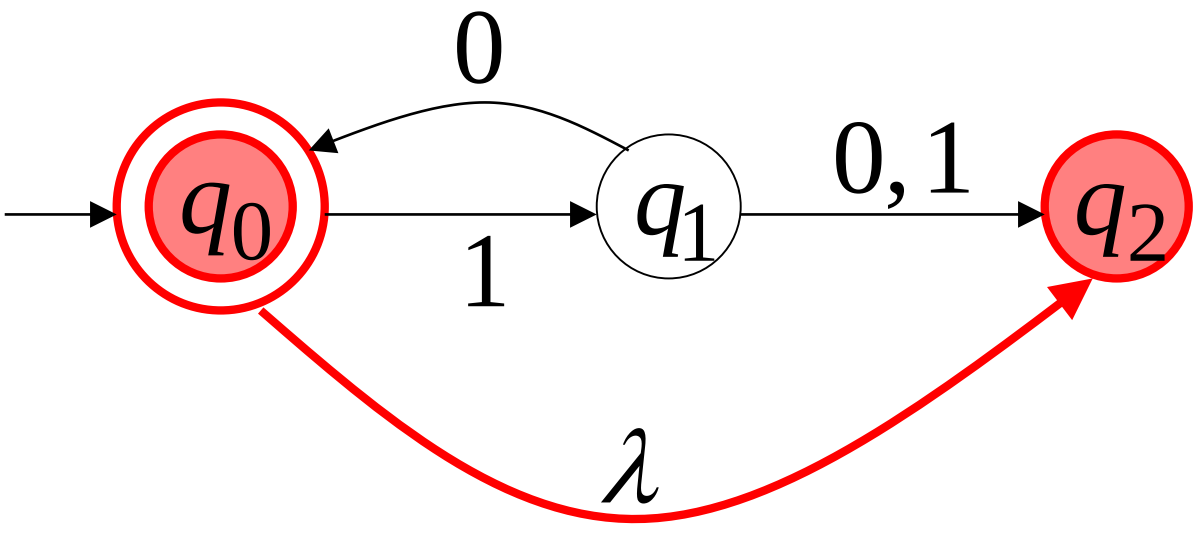
$$\delta(q_0, 1) = \{q_1\}$$



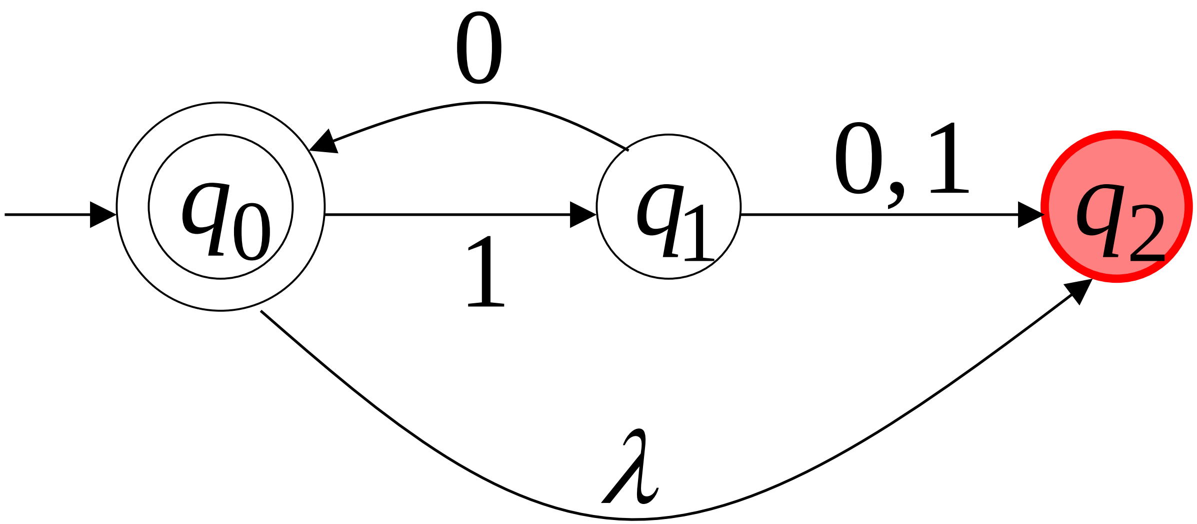
$$\delta(q_1, 0) = \{q_0, q_2\}$$



$$\delta(q_0, \lambda) = \{q_2\}$$



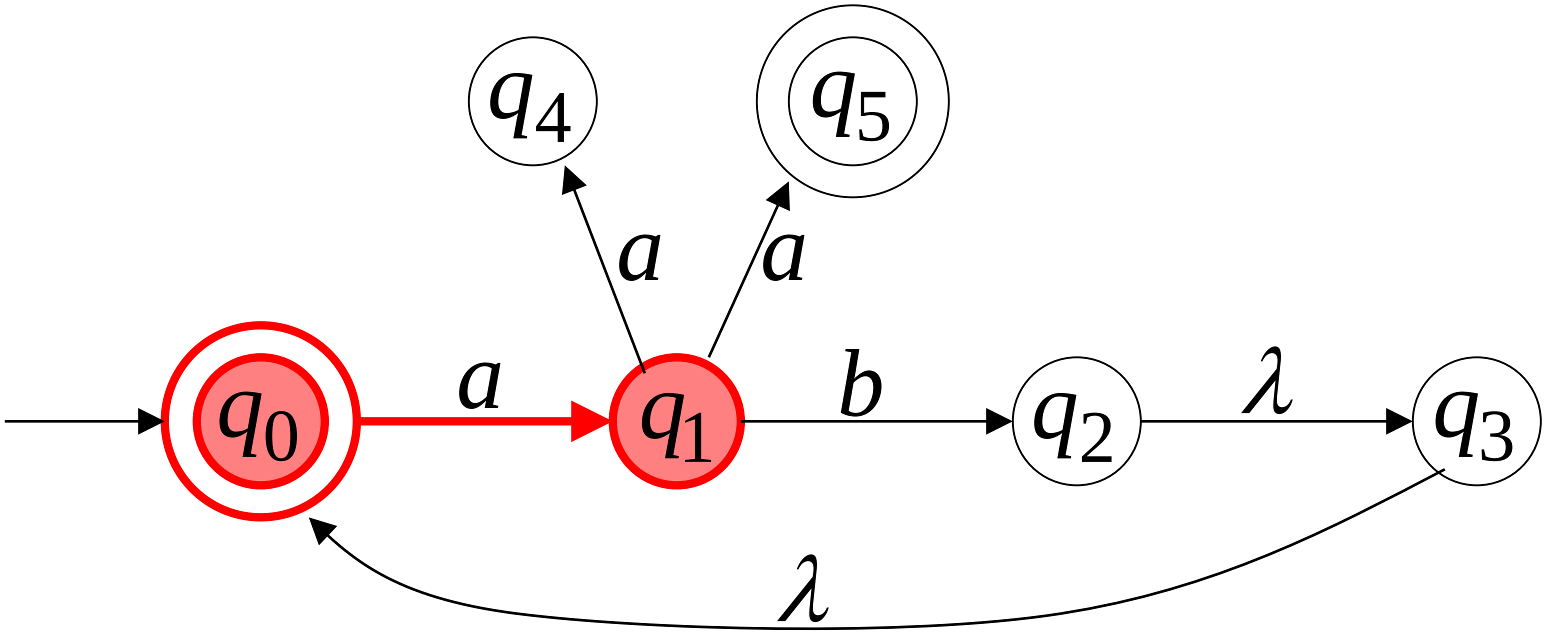
$$\delta(q_2, 1) = \varnothing$$



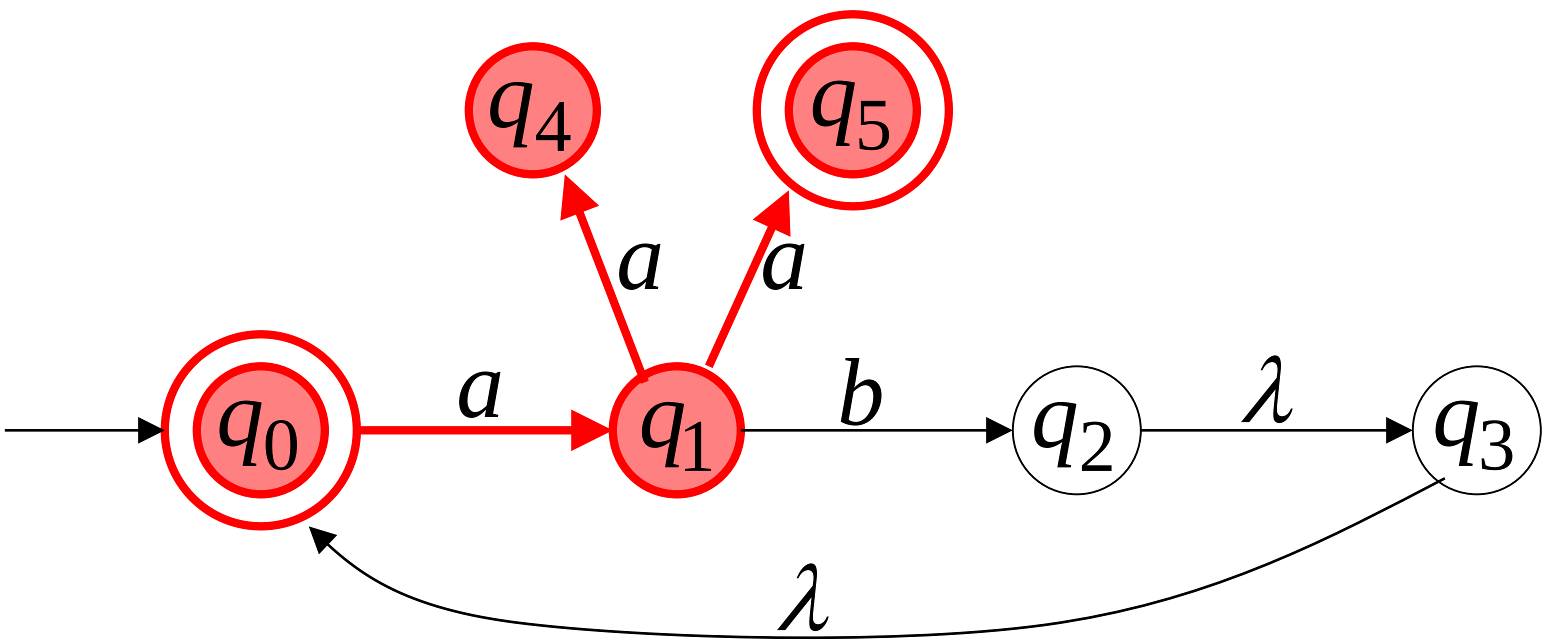
Extended Transition Function δ^*

Same with δ but applied on strings

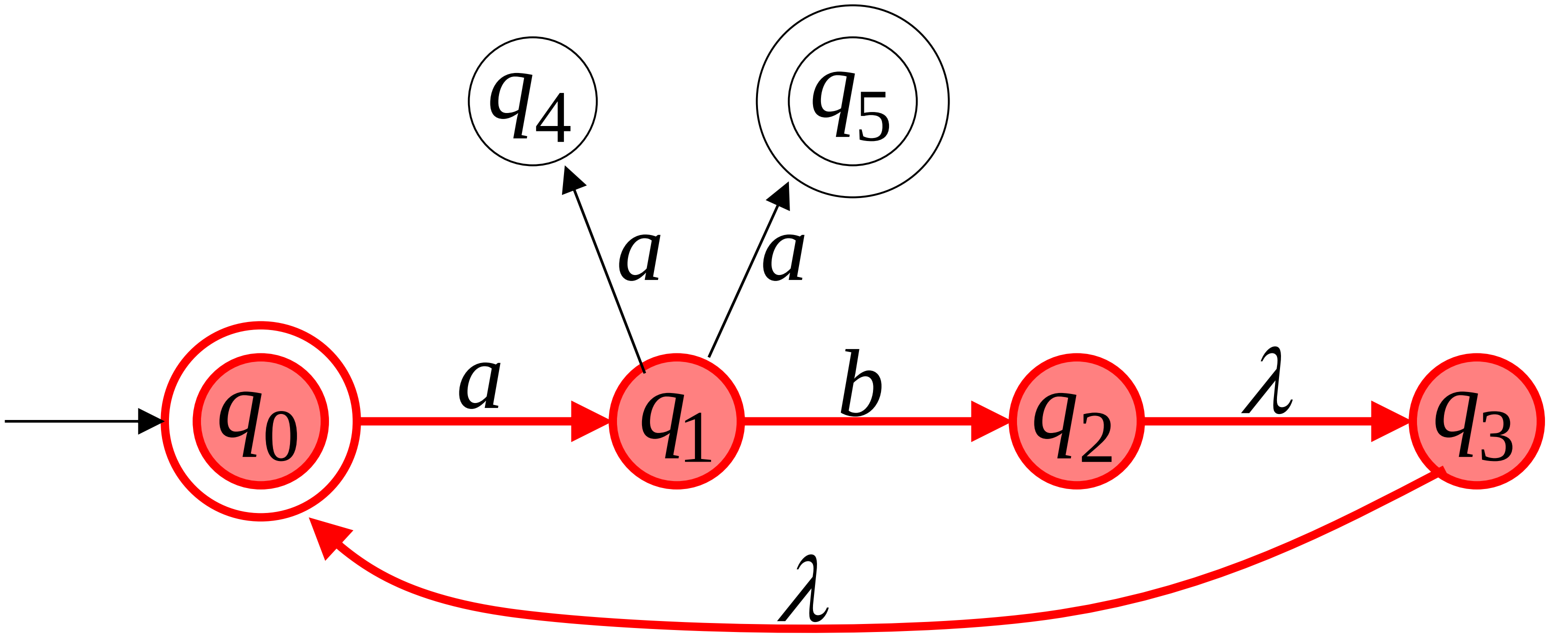
$$\delta^*(q_0, a) = \{q_1\}$$



$$\delta^*(q_0, aa) = \{q_4, q_5\}$$



$$\delta^*(q_0, ab) = \{q_2, q_3, q_0\}$$



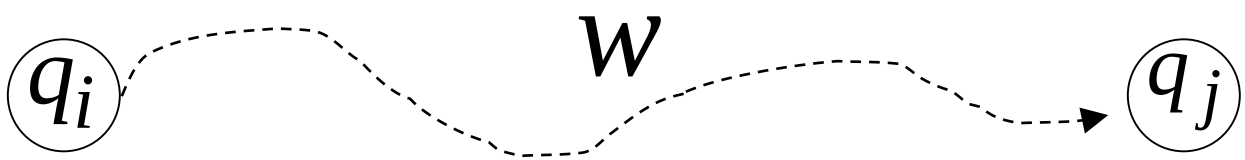
Special case:

for any state q

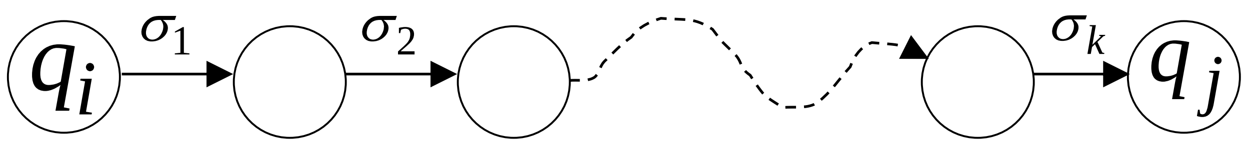
$$q \in \delta^*(q, \lambda)$$

In general

$q_j \in \delta^*(q_i, w)$: there is a walk from q_i to q_j with label w



$$w = \sigma_1 \sigma_2 \cdots \sigma_k$$



The Language of an NFA M

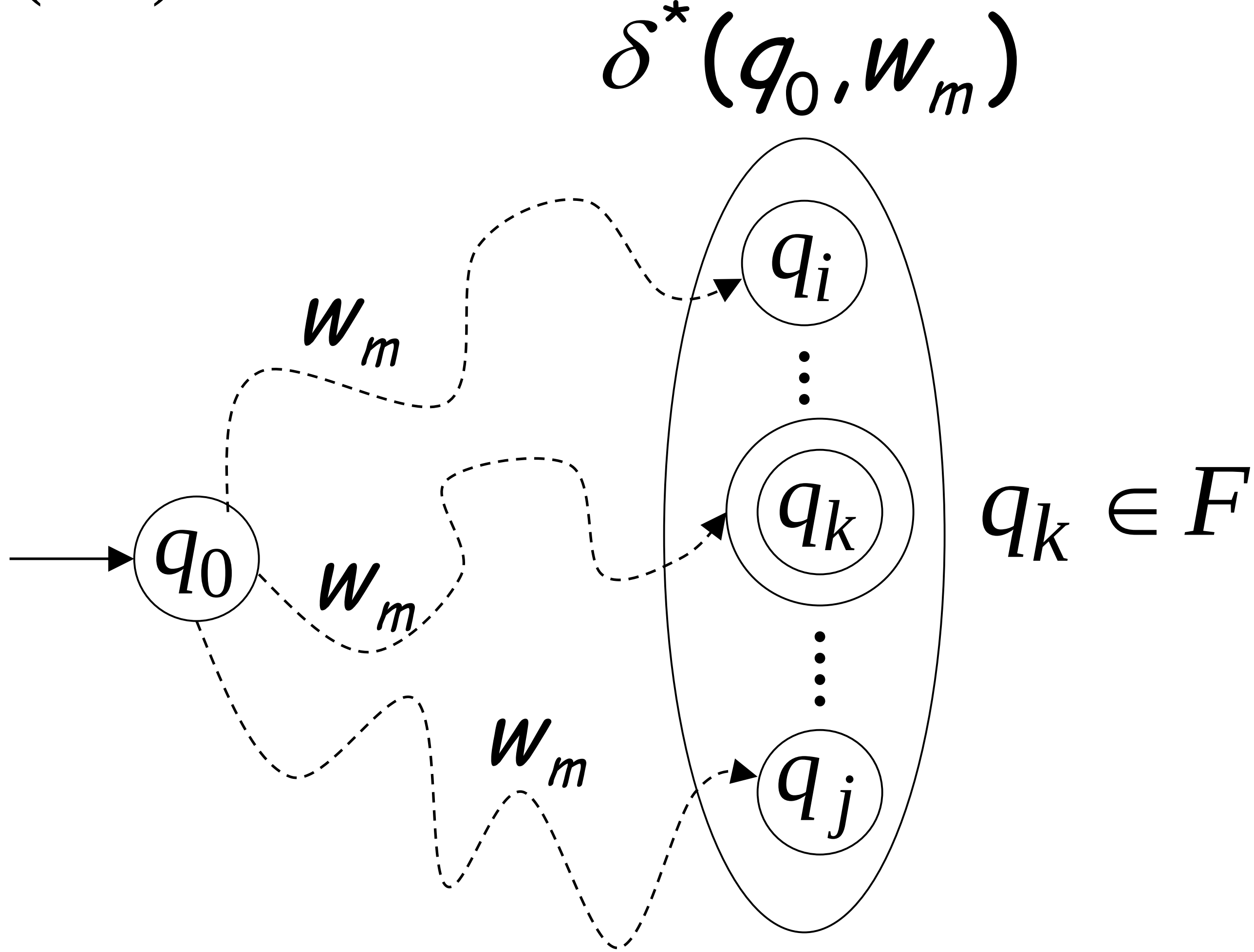
The language accepted by M is:

$$L(M) = \{w_1, w_2, \ldots w_n\}$$

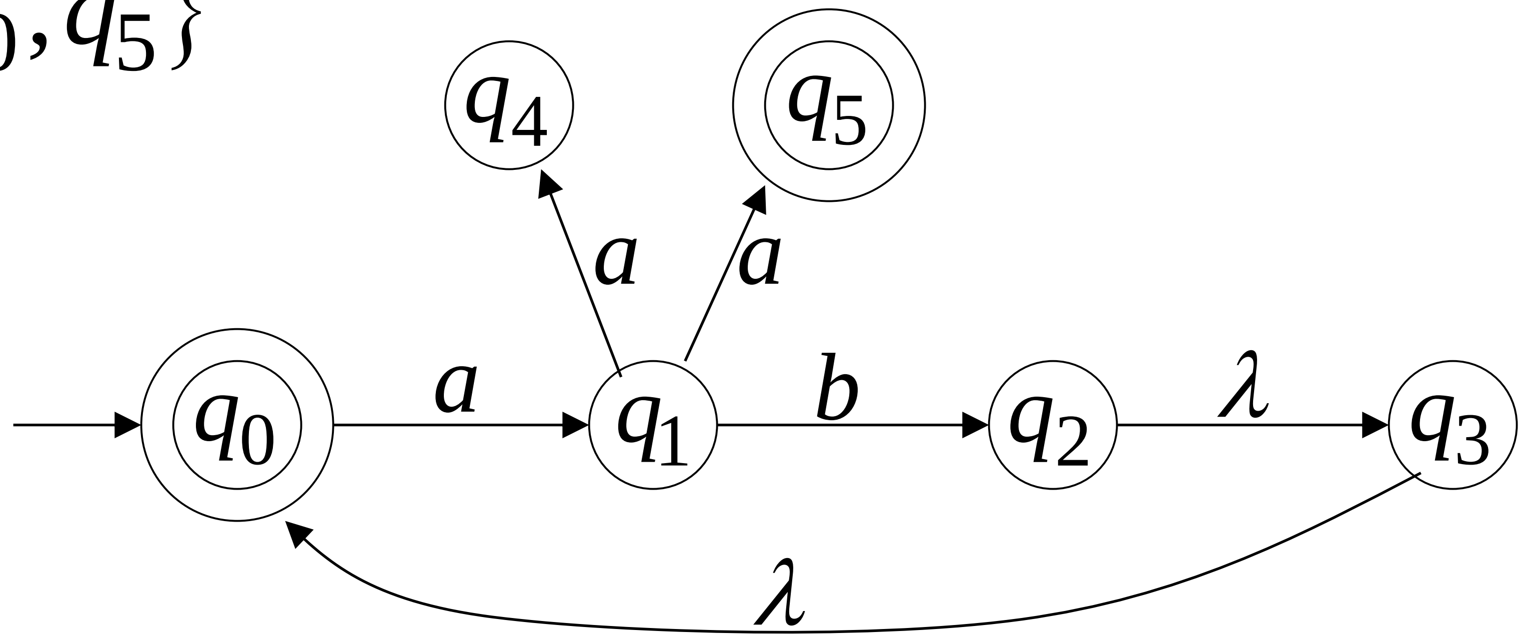
where $\delta^*(q_0, w_m) = \{q_i, \ldots, q_k, \ldots, q_j\}$

and there is some $q_k \in F$ (accepting state)

$$w_m \in L(M)$$



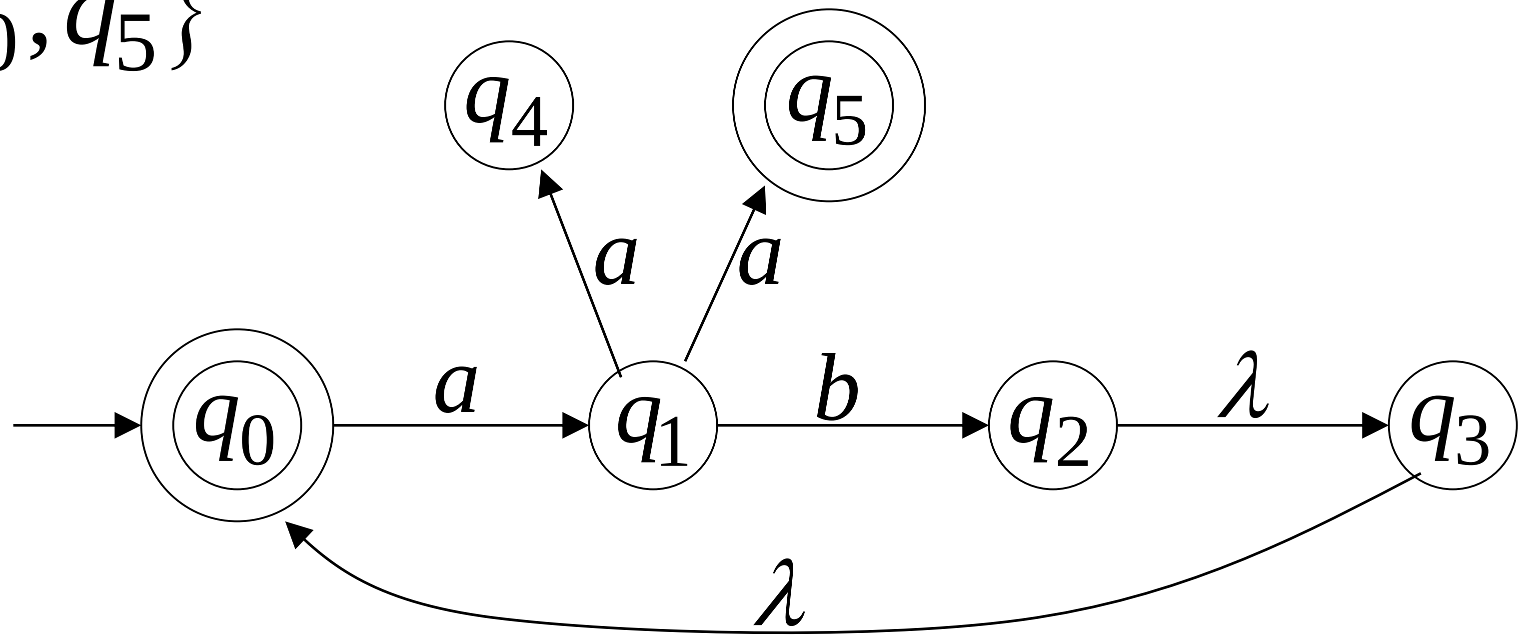
$$F = \{q_0, q_5\}$$



$$\delta^*(q_0, aa) = \{q_4, \underline{q_5}\} \Rightarrow aa \in L(M)$$

$$\underline{q_5} \in F$$

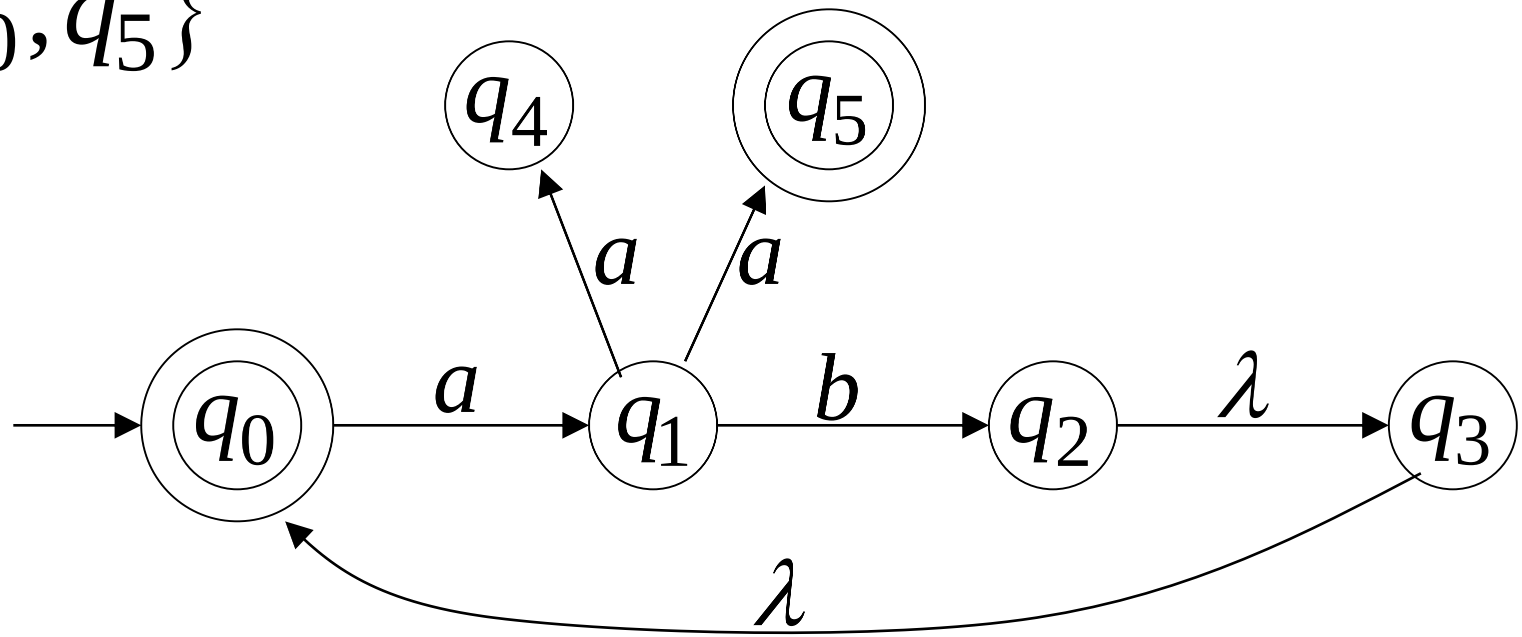
$$F = \{q_0, q_5\}$$



$$\delta^*(q_0, ab) = \{q_2, q_3, \underline{q_0}\} \Rightarrow ab \in L(M)$$

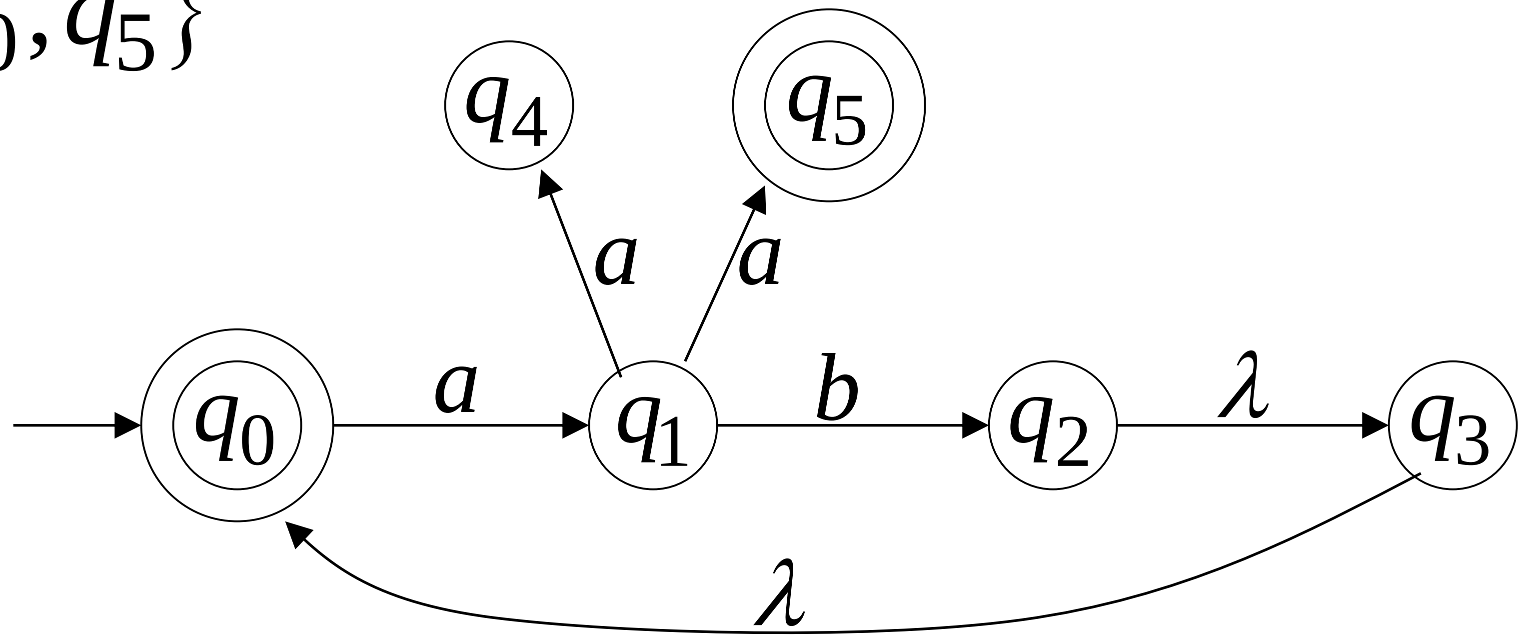
$$\in F$$

$$F = \{q_0, q_5\}$$



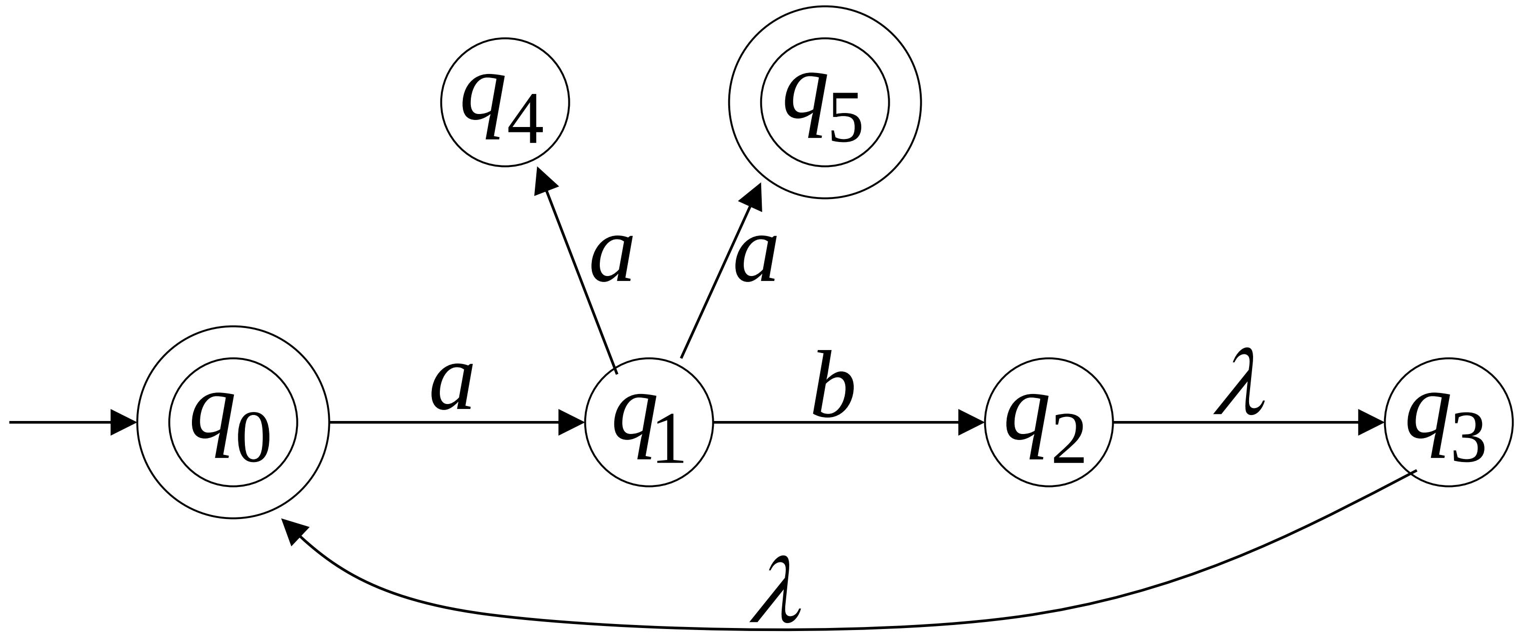
$$\delta^*(q_0, abaa) = \{q_4, \underbrace{q_5}_{\in F}\} \Rightarrow aaba \in L(M)$$

$$F = \{q_0, q_5\}$$



$$\delta^*(q_0, aba) = \{q_1\} \quad \Longrightarrow \quad aba \notin L(M)$$

$$\notin F$$



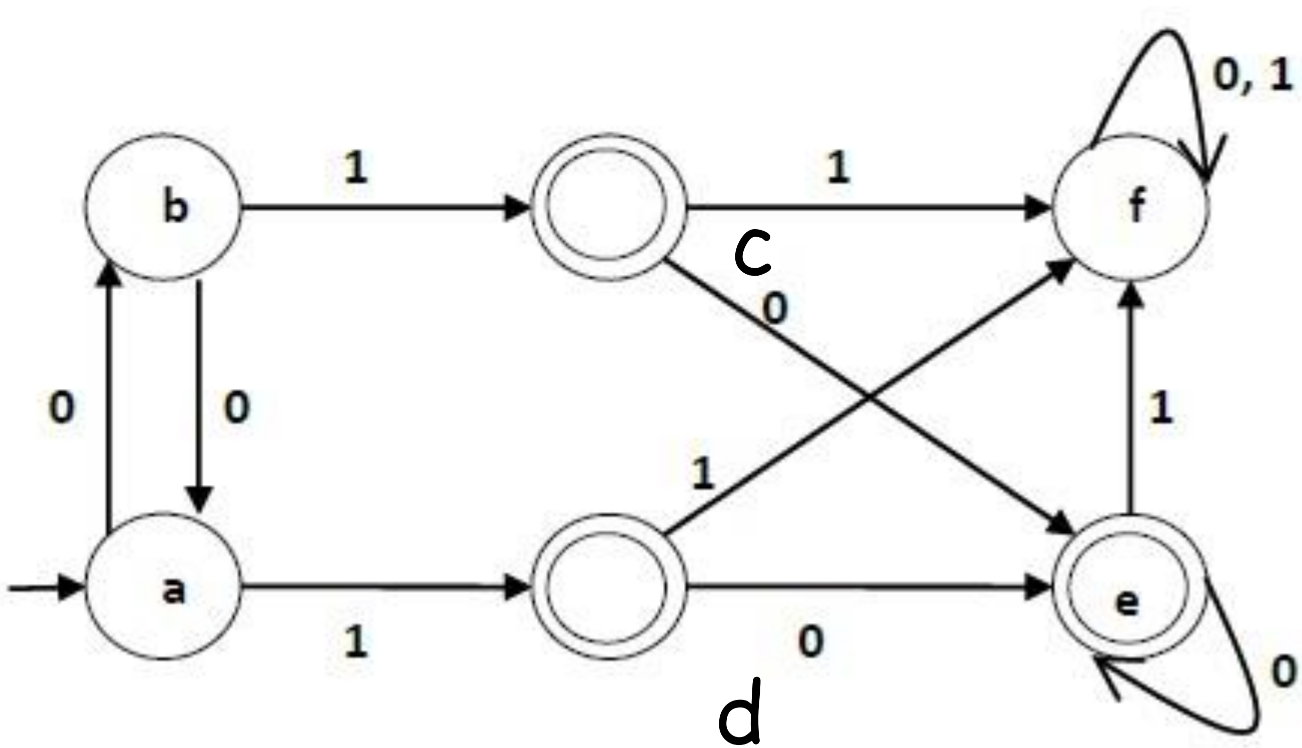
$$L(M) = \{ab\}^* \cup \{ab\}^* \{aa\}$$

Minimization of FA

- **Input** – DFA
- **Output** – Minimized DFA
- **Step 1** – Draw a table for all pairs of states (Qi, Qj) [All are unmarked initially]
- **Step 2** – Consider every state pair (Qi, Qj) in the DFA where Qi ∈ F and Qj ∉ F or vice versa and mark them. [Here F is the set of final states]
- **Step 3** – Repeat this step until we cannot mark anymore states –

  If there is an unmarked pair (Qi, Qj), mark it if the pair {δ (Qi, A), δ (Qi, A)} is marked for some input alphabet.
- **Step 4** – Combine all the unmarked pair (Qi, Qj) and make them a single state in the reduced DFA.

Example

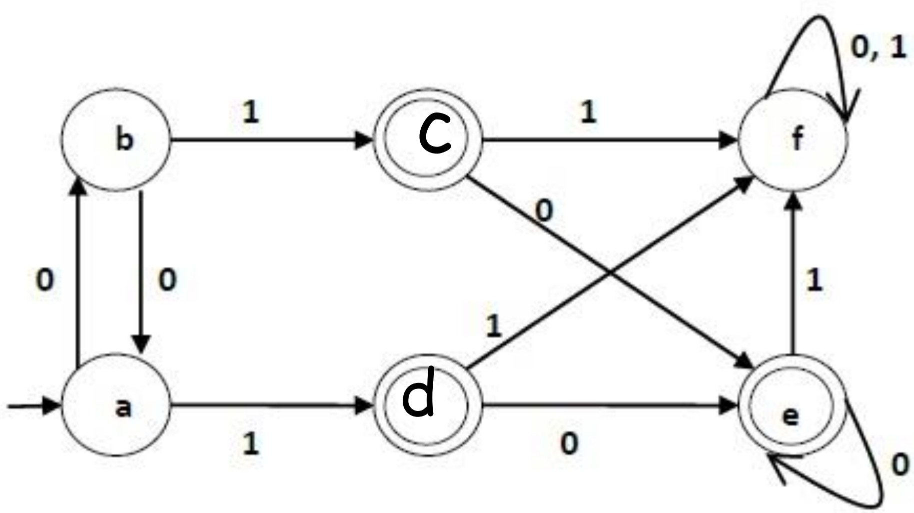


- **Step 1** – Draw a table for all pairs of states (Qi, Qj) [All are unmarked initially].

a						
b						
c						
d						
e						
f						
	a	b	c	d	e	F

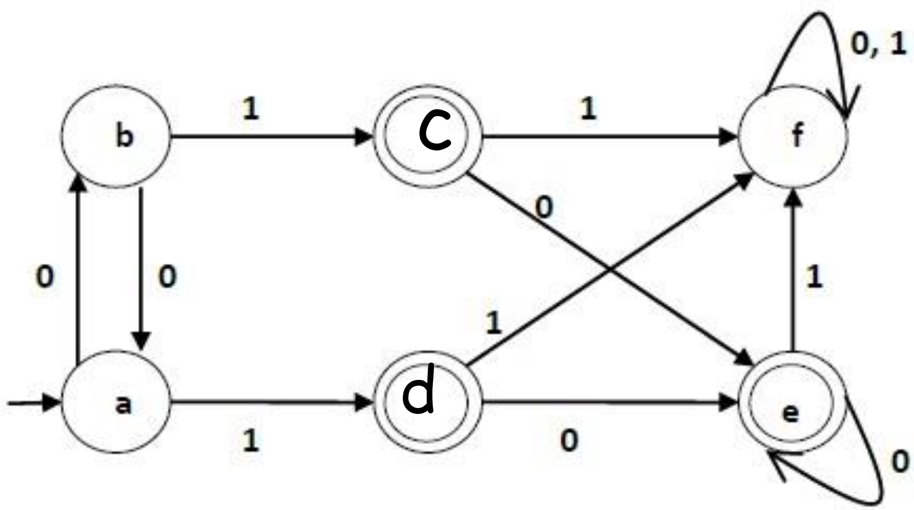
- **Step 1** – Draw a table for all pairs of states (Qi, Qj) [All are unmarked initially].

a						
b						
c						
d						
e						
f						
	a	b	c	d	e	F



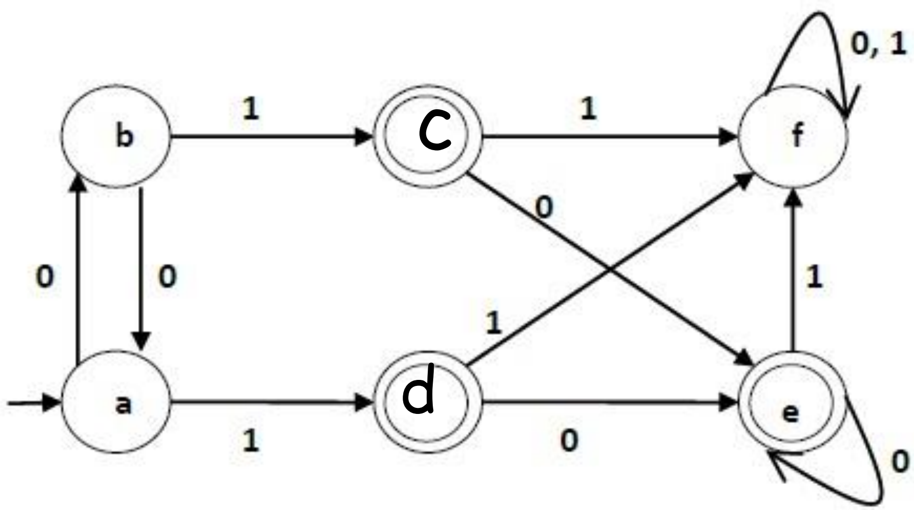
- **Step 2** – Consider every state pair (Qi, Qj) in the DFA where Qi ∈ F and Qj ∉ F or vice versa and mark them. [Here F is the set of final states] If both the states from pair are final then don't mark them.

a						
b						
c						
d						
e						
f						
	a	b	c	d	e	F



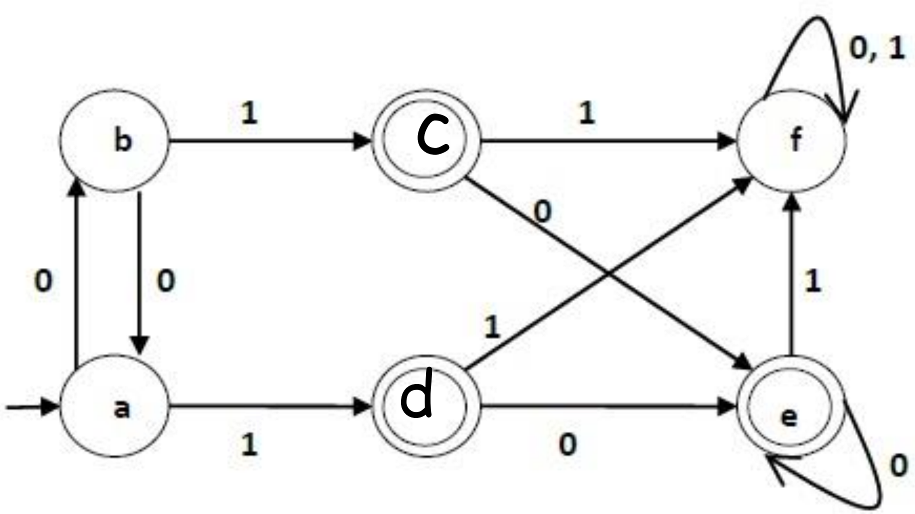
- **Step 2** – Consider every state pair (Qi, Qj) in the DFA where Qi ∈ F and Qj ∉ F or vice versa and mark them. [Here F is the set of final states] . If both the states from pair are final then don't mark them.

a						
b						
c	1	1				
d	1	1				
e	1	1				
f			1	1	1	
	a	B	c	D	e	f



- Repeat this step until we cannot mark anymore states –
  If there is an unmarked pair (Qi, Qj), mark it if the pair {δ (Qi, A), δ (Qi, A)} is marked for some input alphabet.

a						
b						
c	1	1				
d	1	1				
e	1	1				
f			1	1	1	
	a	b	c	d	e	f



Unmarked pair is (a,f)

$\delta(a,0) = b$
$\delta(f,0) = f$

(b,f)

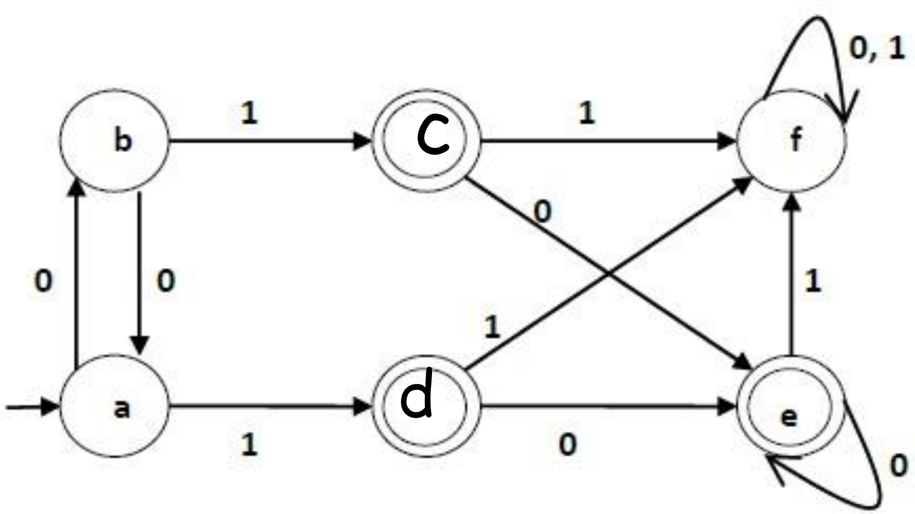
Resultant pair are marked

$\delta(a,1) = d$
$\delta(f,1) = f$

(d,f)

- Repeat this step until we cannot mark anymore states –
  If there is an unmarked pair (Qi, Qj), mark it if the pair {δ (Qi, A), δ (Qi, A)} is marked for some input alphabet.

a						
b						
c	1	1				
d	1	1				
e	1	1				
f	2		1	1	1	
	a	b	c	d	e	f



Unmarked pair is (a,f)

δ (a,0)= b
δ (f,0)= f

(b,f)

Resultant pair are marked

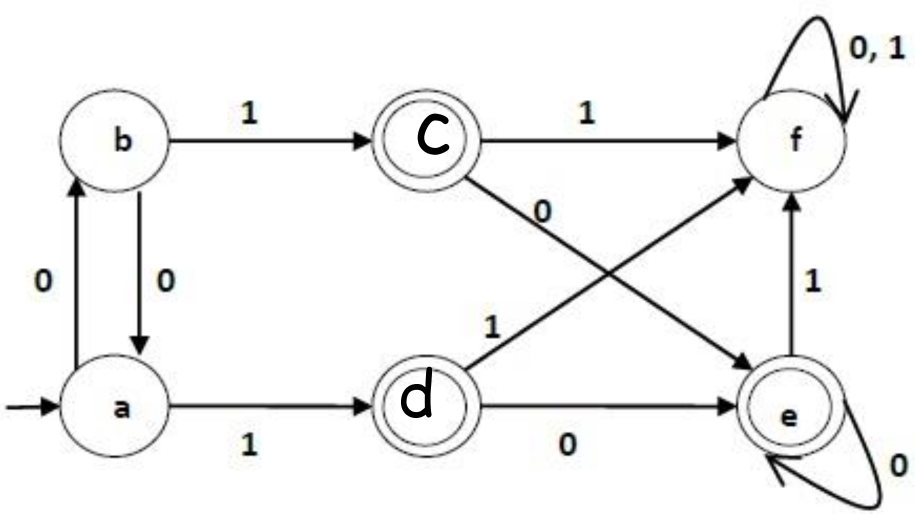
δ (a,1)= d
δ (f,1)= f

(d,f)

- Repeat this step until we cannot mark anymore states –

  If there is an unmarked pair (Qi, Qj), mark it if the pair {δ (Qi, A), δ (Qi, A)} is marked for some input alphabet.

a						
b						
c	1	1				
d	1	1				
e	1	1				
f	2		1	1	1	
	a	b	c	d	e	f



Unmarked pair is (a,b)

δ (a,0)= b
δ (b,0)= a

(b,a)

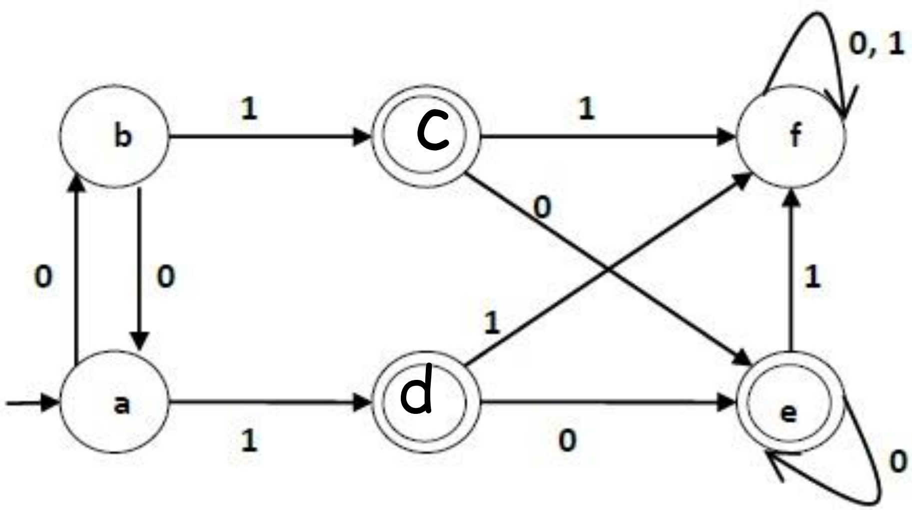
Resultant pair are unmarked

δ (a,1)= d
δ (b,1)= c

(d,c)

- Repeat this step until we cannot mark anymore states –
  If there is an unmarked pair (Qi, Qj), mark it if the pair {δ (Qi, A), δ (Qi, A)} is marked for some input alphabet.

a						
b						
c	1	1				
d	1	1				
e	1	1				
f	2	2	1	1	1	
	a	b	c	d	e	f



Unmarked pair is (b,f)

$\delta\ (b,0) = a$
$\delta\ (f,0) = f$

(a,f)

Resultant pair are marked

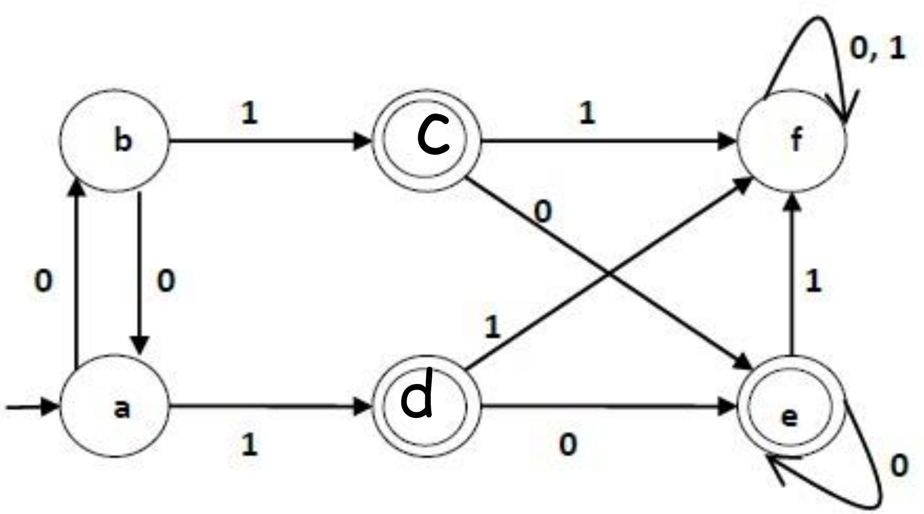
$\delta\ (b,1) = c$
$\delta\ (f,1) = f$

(c,f)

- Repeat this step until we cannot mark anymore states –

  If there is an unmarked pair (Qi, Qj), mark it if the pair {δ (Qi, A), δ (Qi, A)} is marked for some input alphabet.

a						
b						
c	1	1				
d	1	1				
e	1	1				
f	2	2	1	1	1	
	a	b	c	d	e	f



Unmarked pair is (c,e)

$\delta(c,0) = e$

$\delta(e,0) = e$

(e,e)

Resultant pair are unmarked

$\delta(c,1) = f$

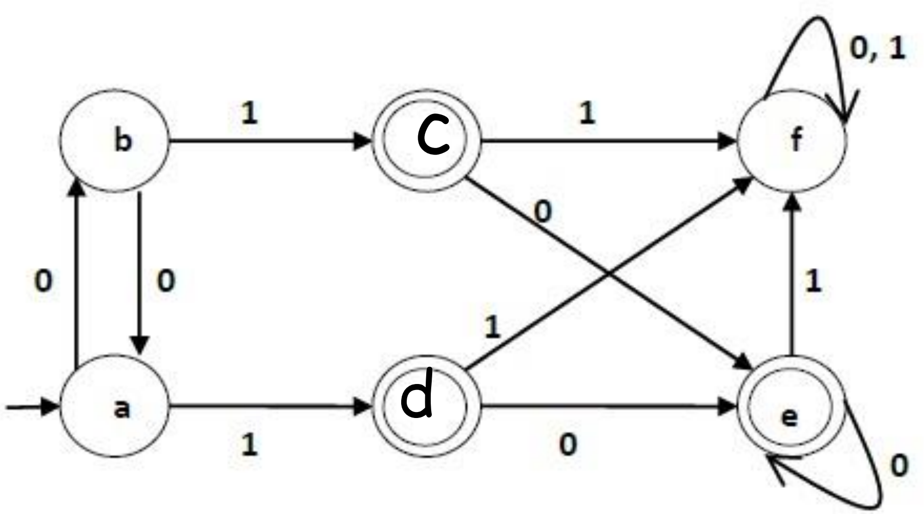
$\delta(e,1) = f$

(f,f)

- Repeat this step until we cannot mark anymore states –

  If there is an unmarked pair (Qi, Qj), mark it if the pair {δ (Qi, A), δ (Qi, A)} is marked for some input alphabet.

a						
b						
c	1	1				
d	1	1				
e	1	1				
f	2	2	1	1	1	
	a	b	c	d	e	f



Unmarked pair is (c,d)

δ (c,0)= e
δ (d,0)= e

(e,e)

Resultant pair are unmarked

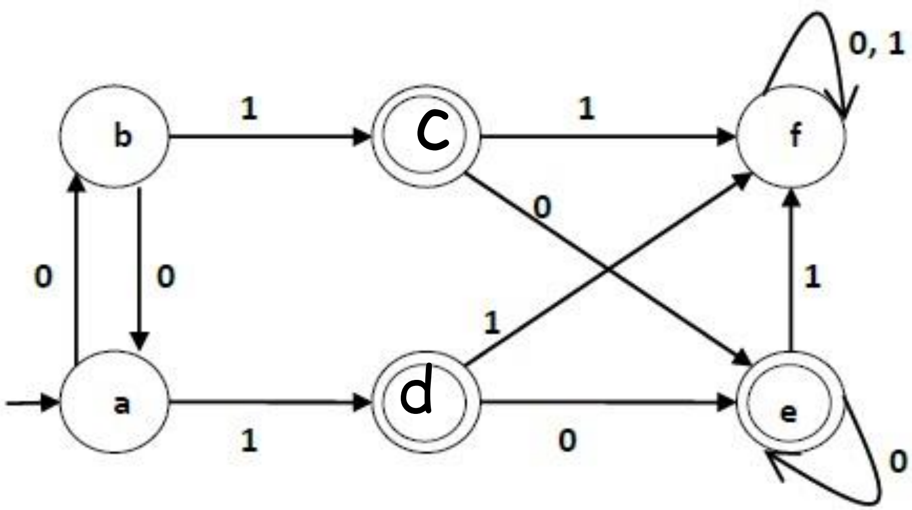
δ (c,1)= f
δ (d,1)= f

(f,f)

- Repeat this step until we cannot mark anymore states –

  If there is an unmarked pair (Qi, Qj), mark it if the pair {δ (Qi, A), δ (Qi, A)} is marked for some input alphabet.

a						
b						
c	1	1				
d	1	1				
e	1	1				
f	2	2	1	1	1	
	a	b	c	d	e	f



Unmarked pair is (d,e)

δ (d,0)= e

δ (e,0)= e

(e,e)

Resultant pair are unmarked

δ (d,1)= f

δ (e,1)= f

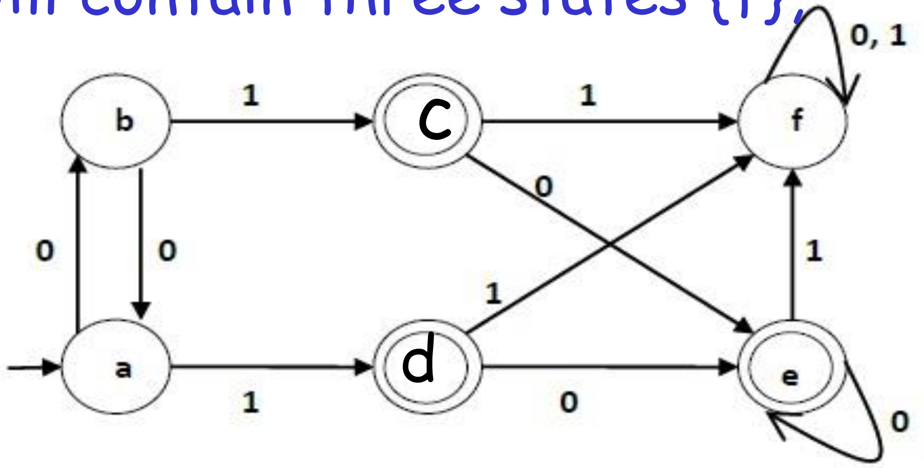
(f,f)

- **Step 4** – Combine all the unmarked pair (Q_i, Q_j) and make them a single state in the reduced DFA.

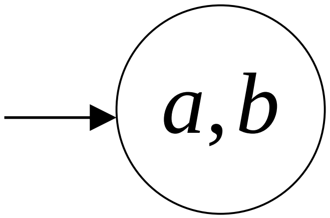
a						
b						
c	1	1				
d	1	1				
e	1	1				
f	2	2	1	1	1	
	a	b	c	d	e	f

- After step 3, we have got state combinations {a, b} {c, d} {c, e} {d, e} that are unmarked.
- We can recombine {c, d} {c, e} {d, e} into {c, d, e}
- Hence we got two combined states as – {a, b} and {c, d, e}

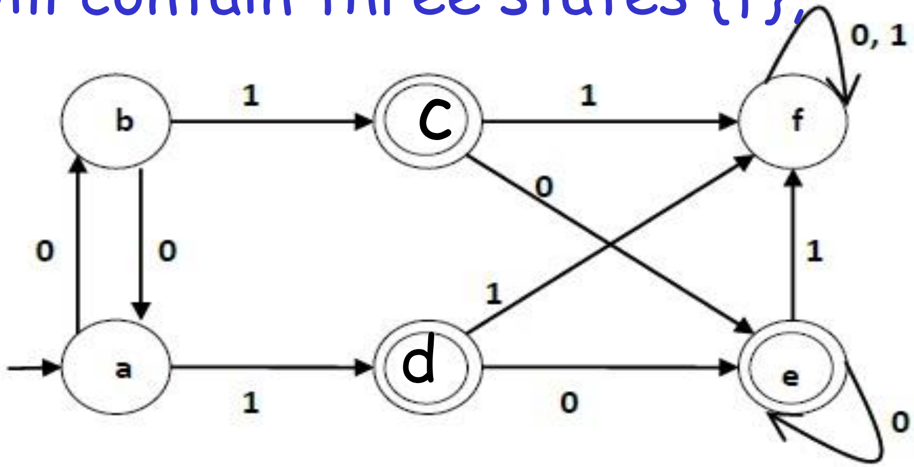
- So the final minimized DFA will contain three states {f}, {a, b} and {c, d, e}



{a, b}



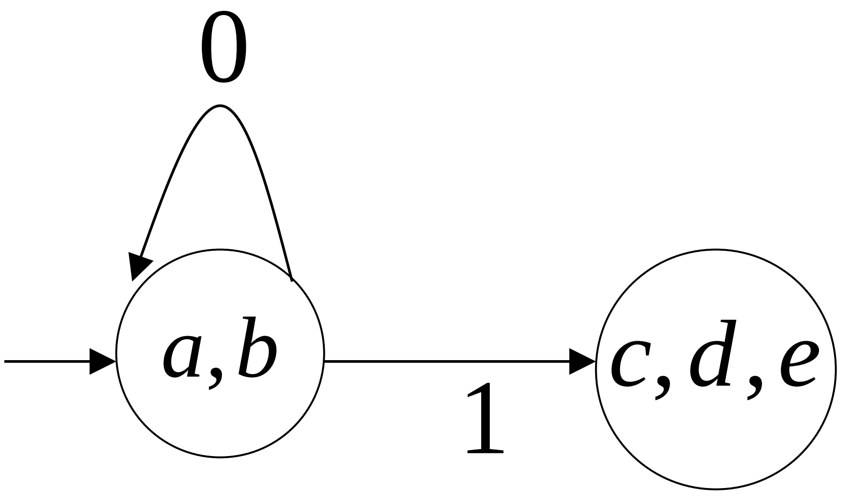
- So the final minimized DFA will contain three states {f}, {a, b} and {c, d, e}



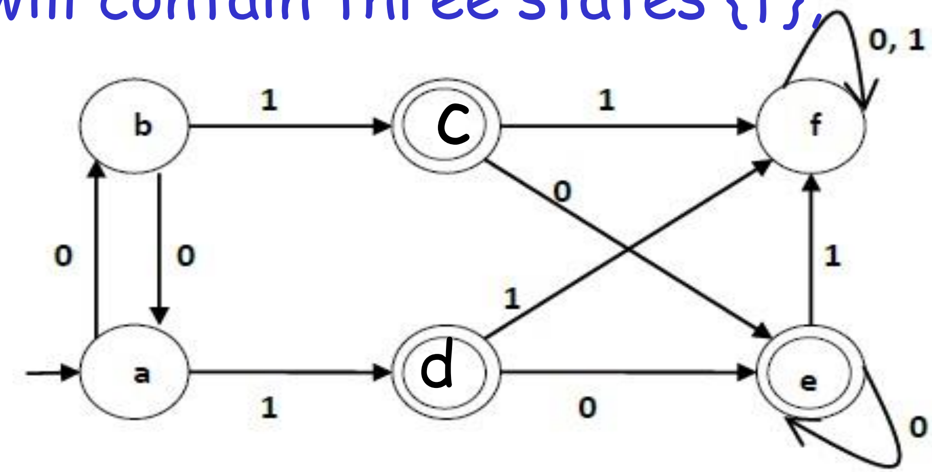
{a, b}

δ (a,0)=b   δ (a,1)=d

δ (b,0)=a   δ (b,1)=c



- So the final minimized DFA will contain three states {f}, {a, b} and {c, d, e}

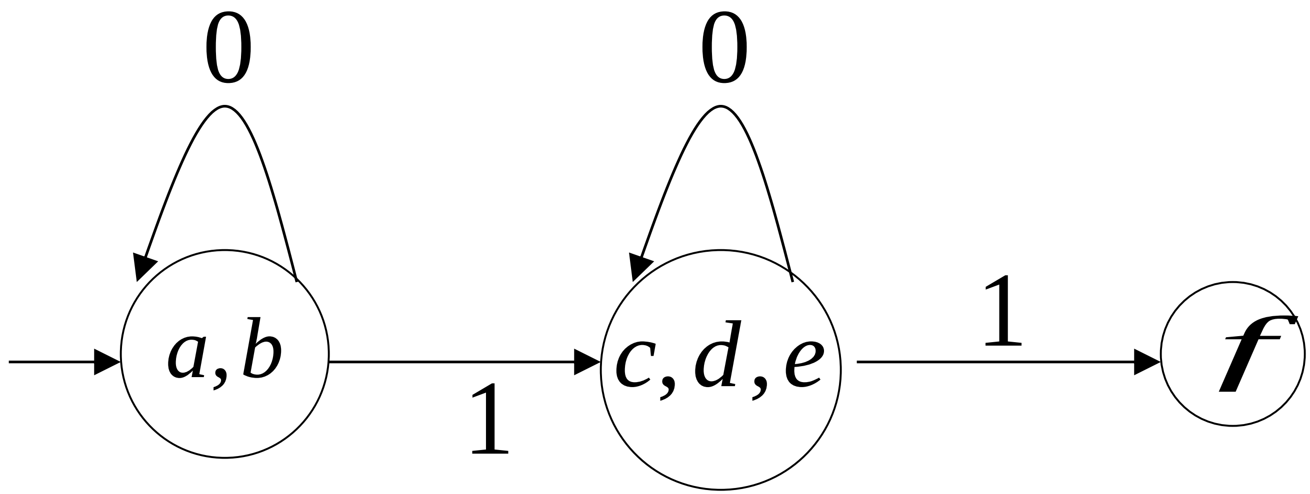


{c, d,e}

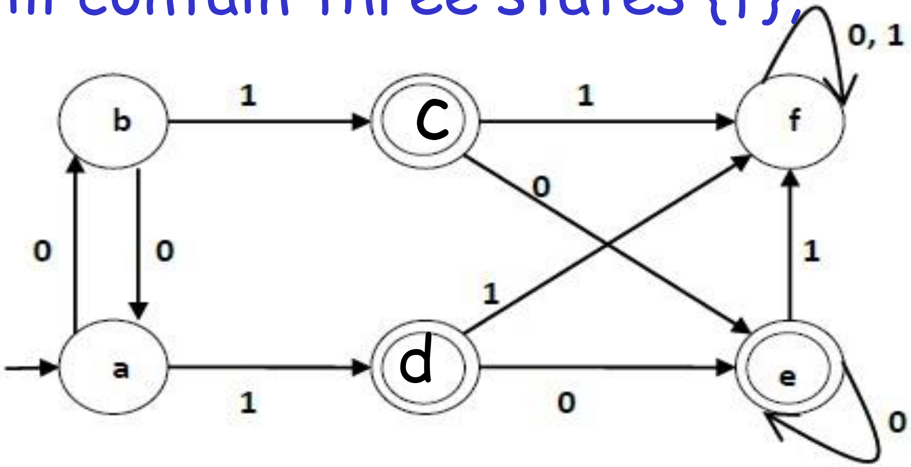
δ (c,0)=e δ (c,1)=f

δ (d,0)=e δ (d,1)=f

δ (e,0)=e δ (e,1)=f

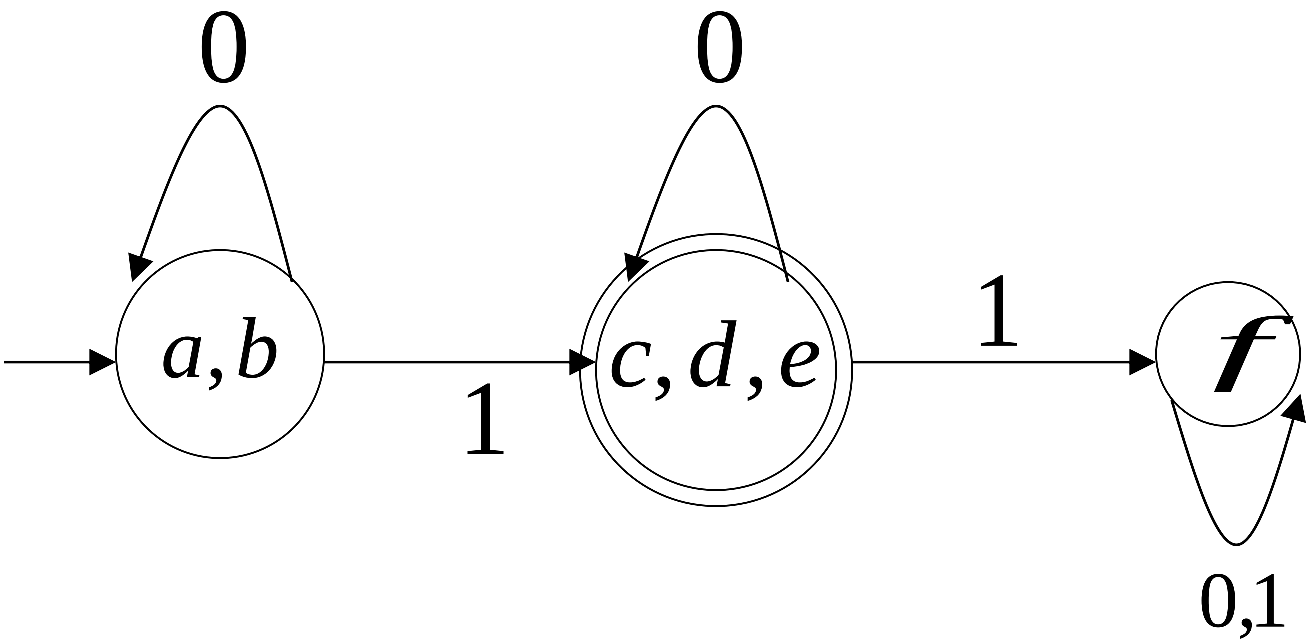


- So the final minimized DFA will contain three states {f}, {a, b} and {c, d, e}

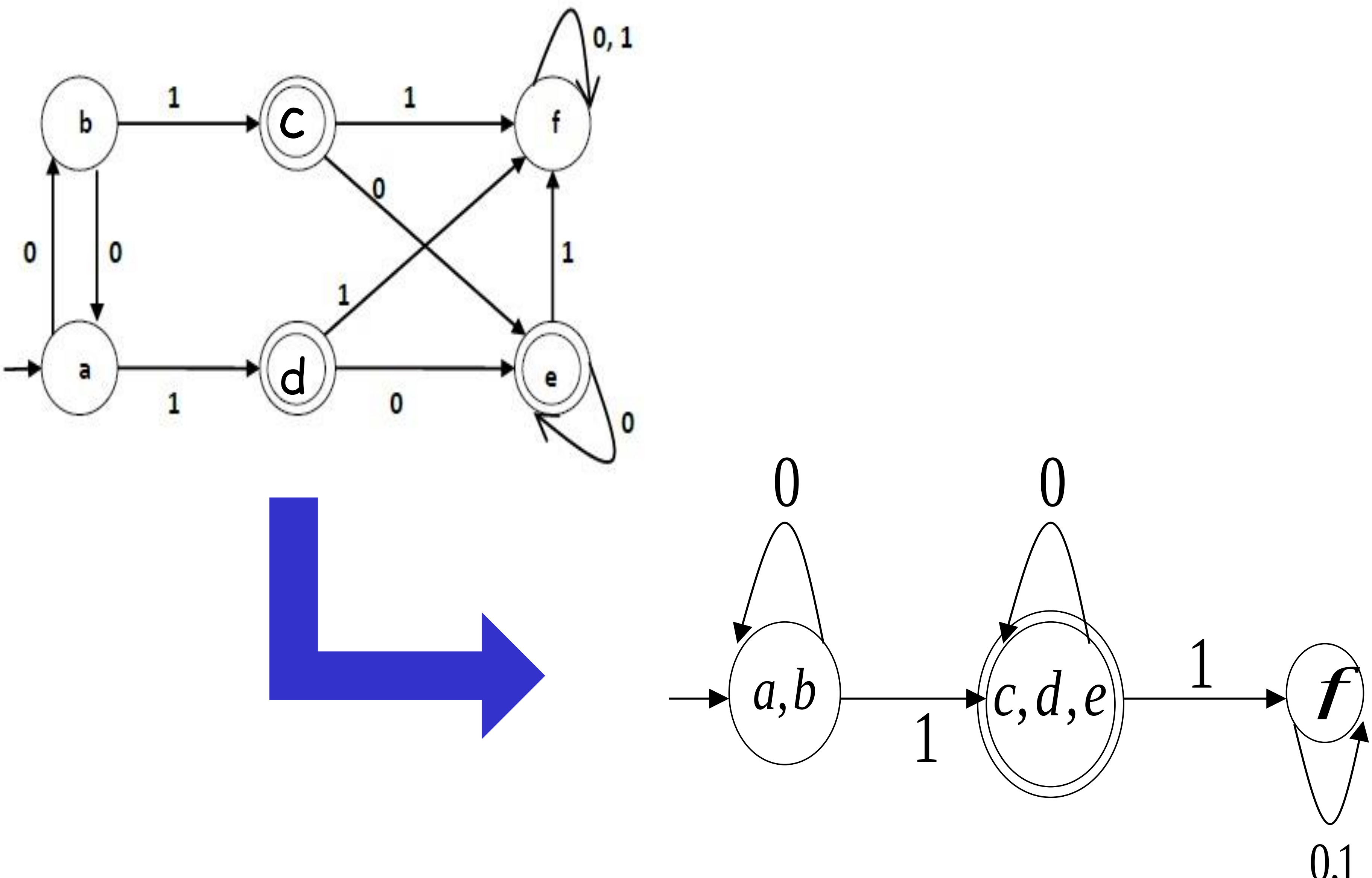


{f}

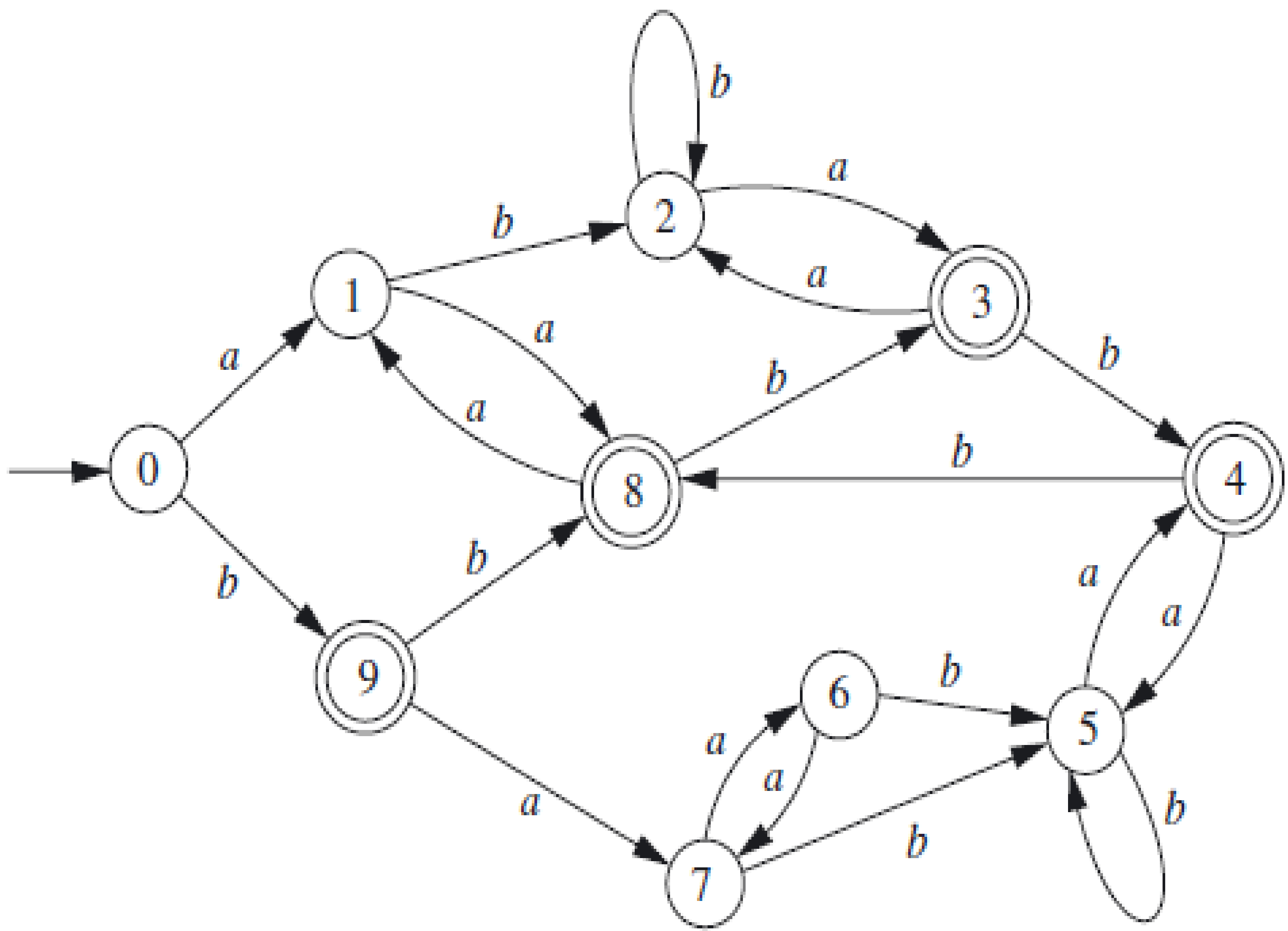
δ (f,0)=f   δ (f,1)=f



Example

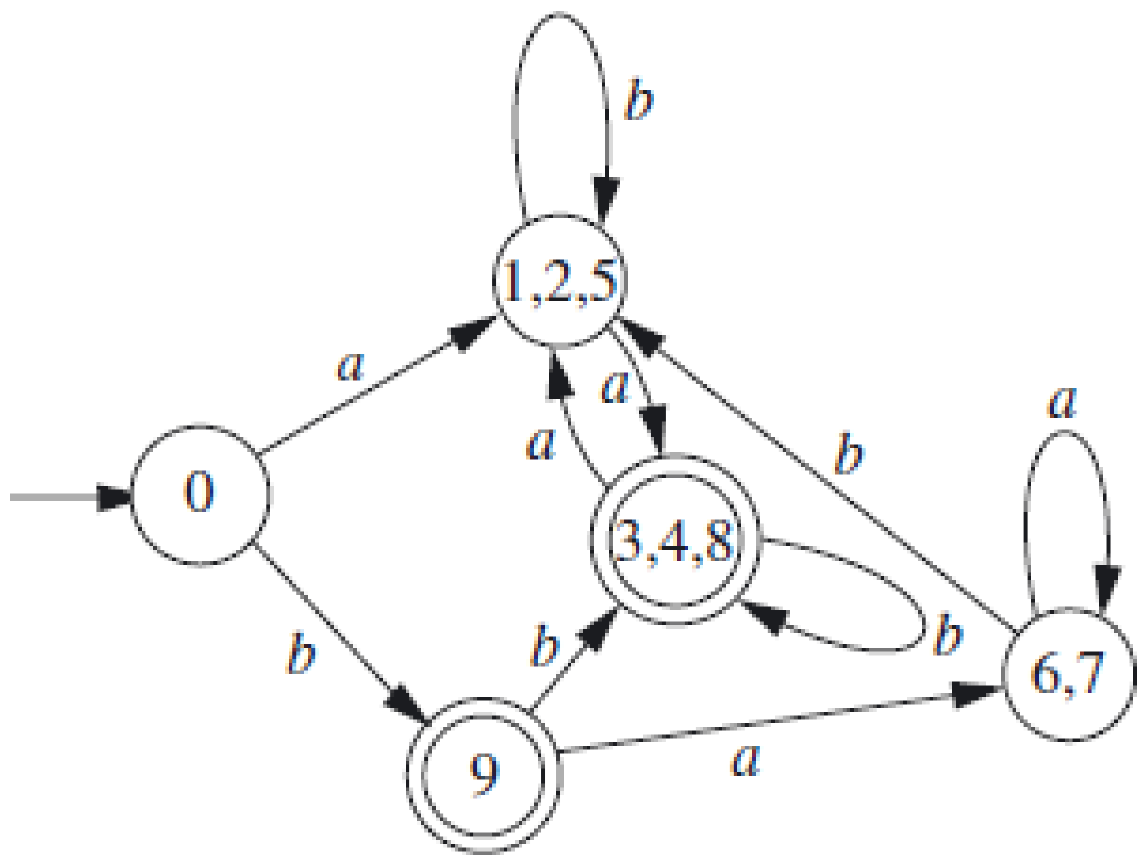


Minimize following DFA

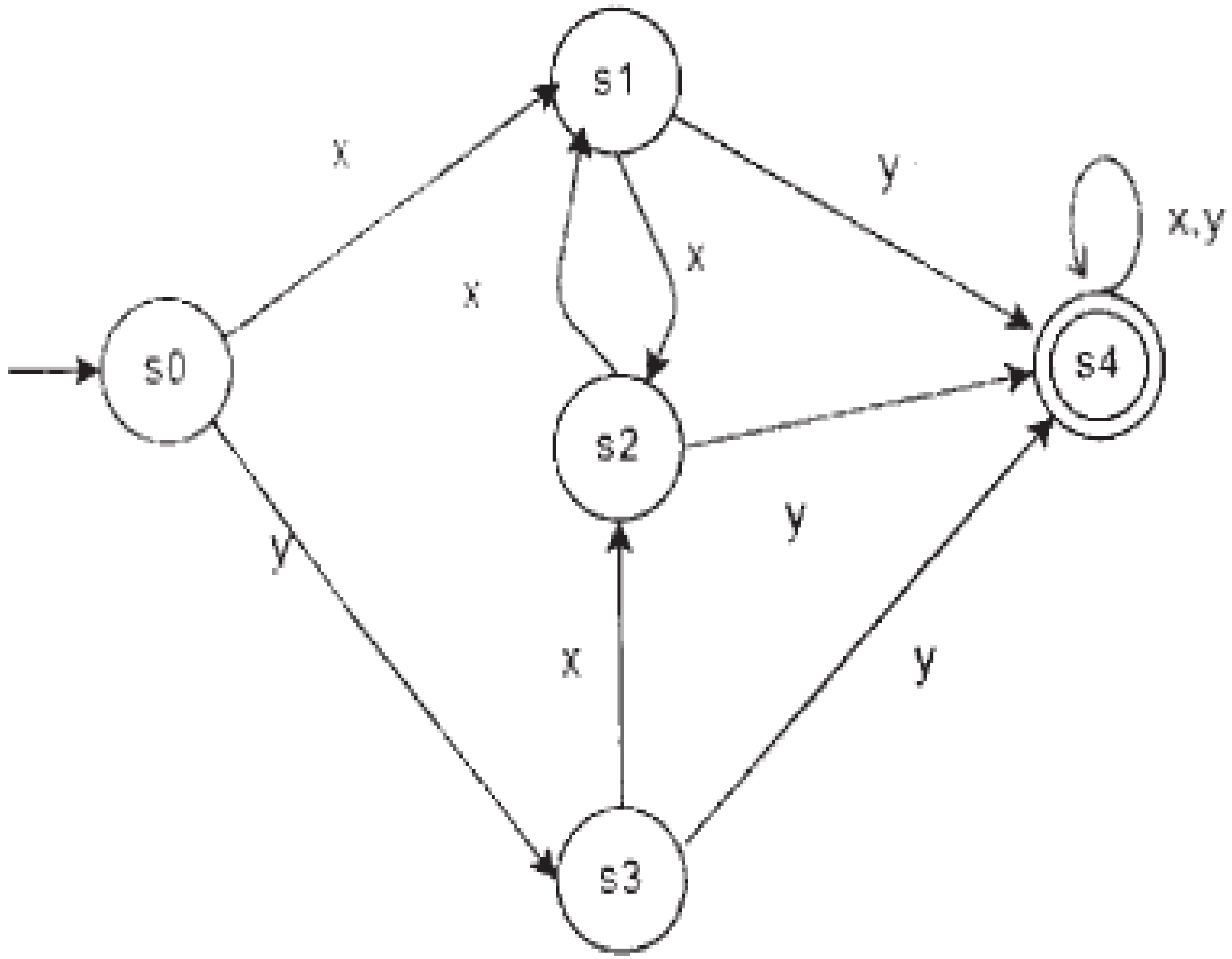


1	2								
2	2								
3	1	1	1						
4	1	1	1						
5	2			1	1				
6	2	2	2	1	1	2			
7	2	2	2	1	1	2			
8	1	1	1			1	1	1	
9	1	1	1	2	3	1	1	1	2
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Final DFA



Minimize the following automata.



NFAs accept the Regular Languages

Equivalence of Machines

Definition:

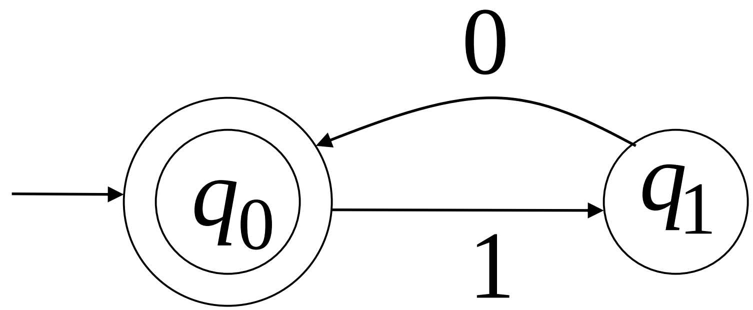
Machine M_1 is equivalent to machine M_2

if $L(M_1) = L(M_2)$

Example of equivalent machines

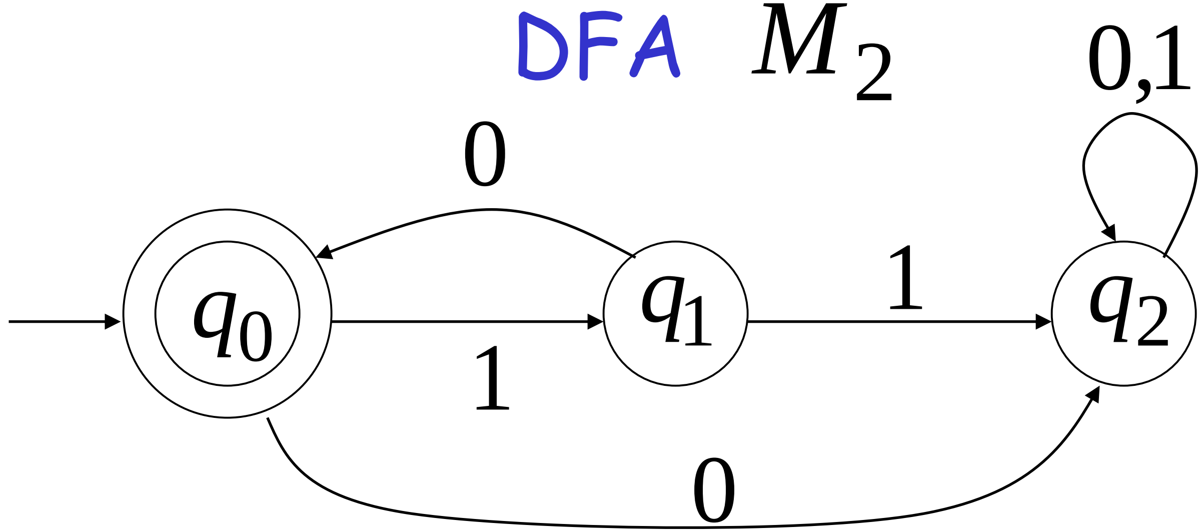
NFA M_1

$$L(M_1) = \{10\}^*$$



DFA M_2

$$L(M_2) = \{10\}^*$$



Theorem:

$$\left\{ \begin{array}{l} \text{Languages} \\ \text{accepted} \\ \text{by NFAs} \end{array} \right\} = \left\{ \begin{array}{l} \text{Regular} \\ \text{Languages} \end{array} \right\}$$

Languages
accepted
by DFAs

NFAs and DFAs have the same computation power, accept the same set of languages

Proof: we only need to show

$$\left\{\begin{array}{l}\text{Languages}\\ \text{accepted}\\ \text{by NFAs}\end{array}\right\} \supseteq \left\{\begin{array}{l}\text{Regular}\\ \text{Languages}\end{array}\right\}$$

AND

$$\left\{\begin{array}{l}\text{Languages}\\ \text{accepted}\\ \text{by NFAs}\end{array}\right\} \subseteq \left\{\begin{array}{l}\text{Regular}\\ \text{Languages}\end{array}\right\}$$

Proof-Step 1

$$\left\{ \begin{array}{l} \text{Languages} \\ \text{accepted} \\ \text{by NFAs} \end{array} \right\} \supseteq \left\{ \begin{array}{l} \text{Regular} \\ \text{Languages} \end{array} \right\}$$

Every DFA is trivially an NFA

$\Downarrow$

Any language L accepted by a DFA is also accepted by an NFA

Proof-Step 2

$$\left\{ \begin{array}{l} \text{Languages} \\ \text{accepted} \\ \text{by NFAs} \end{array} \right\} \subseteq \left\{ \begin{array}{l} \text{Regular} \\ \text{Languages} \end{array} \right\}$$

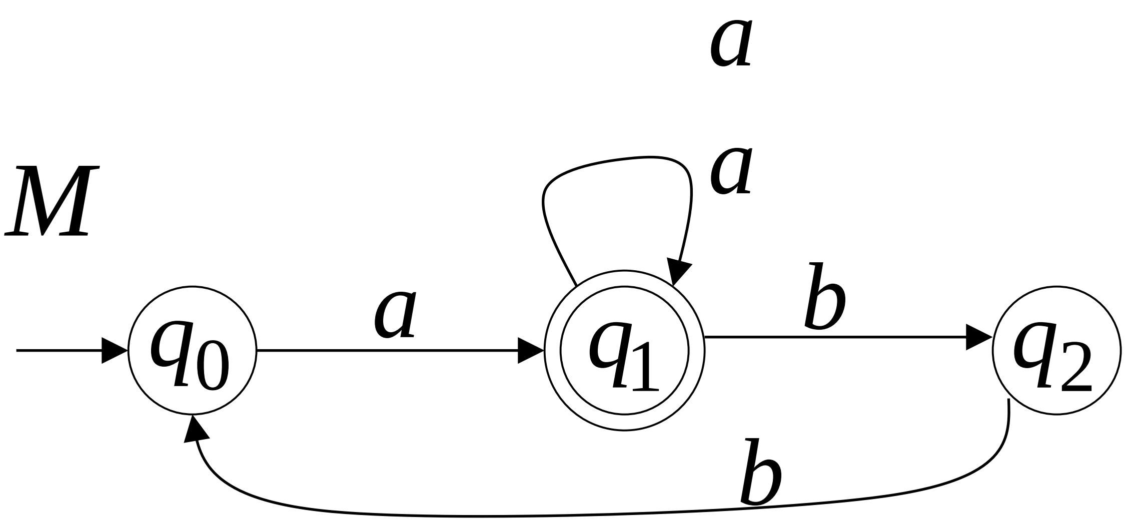
Any NFA can be converted to an equivalent DFA

$$\Downarrow$$

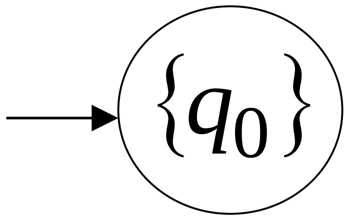
Any language L accepted by an NFA is also accepted by a DFA

Conversion NFA to DFA

NFA M

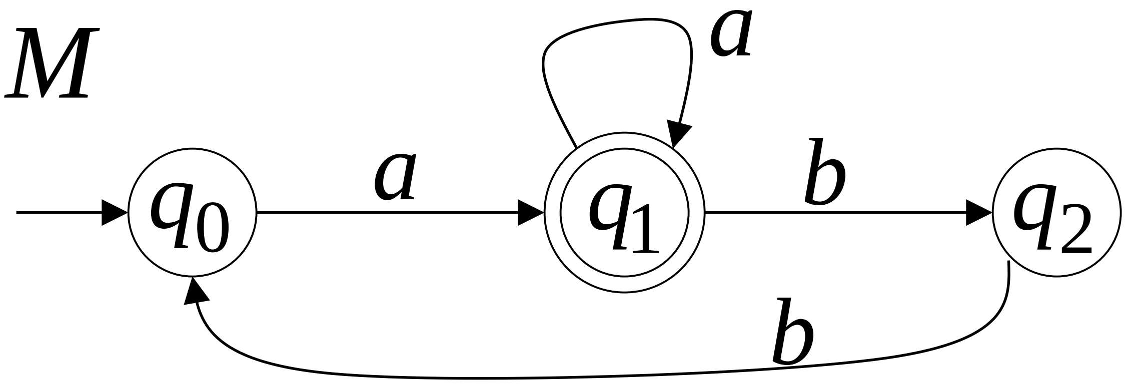


DFA M'

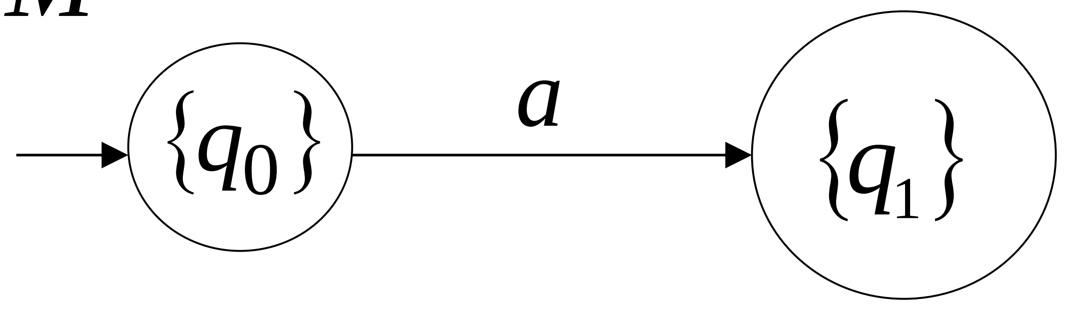


$$\delta^*(q_0, a) = \{q_1\}$$

NFA M

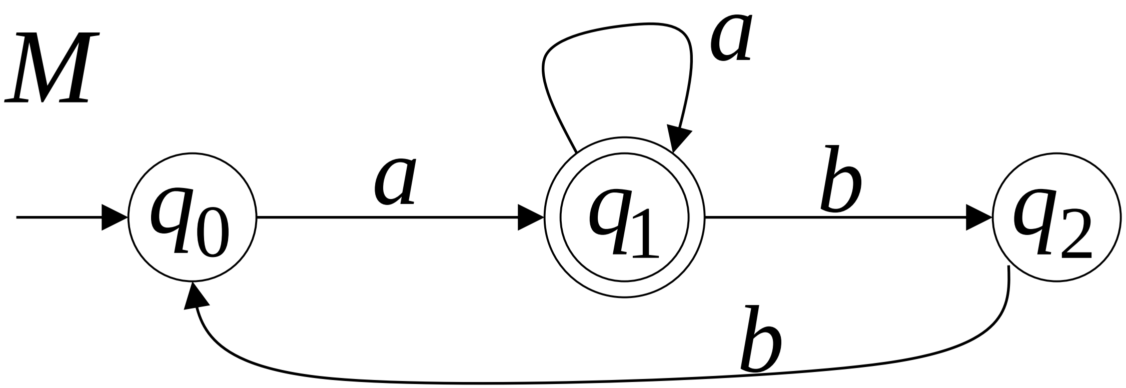


DFA M'

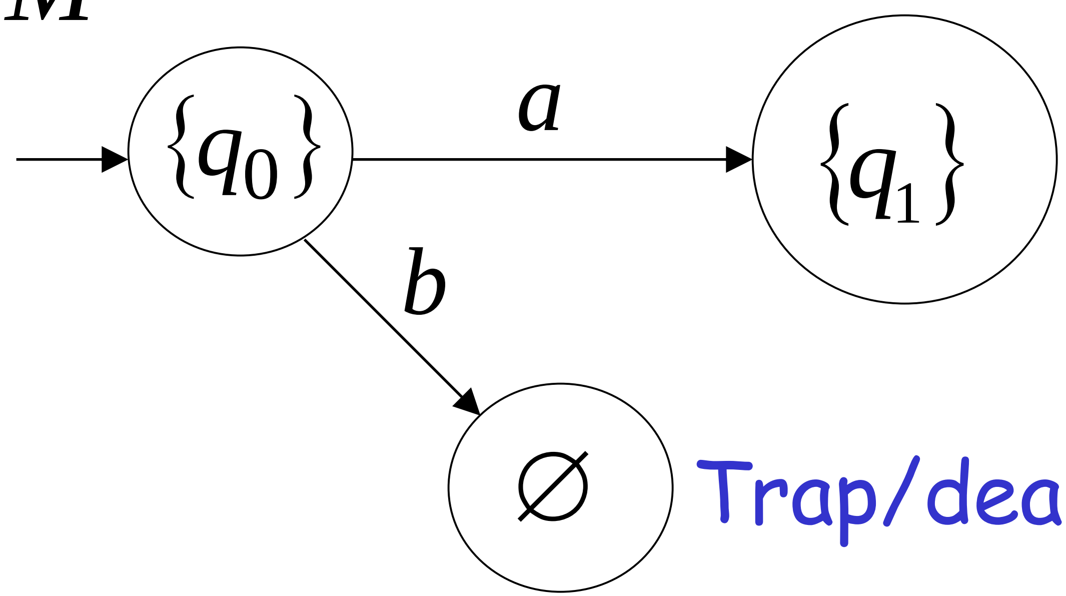


$$\delta^*(q_0, b) = \varnothing$$ empty set

NFA M

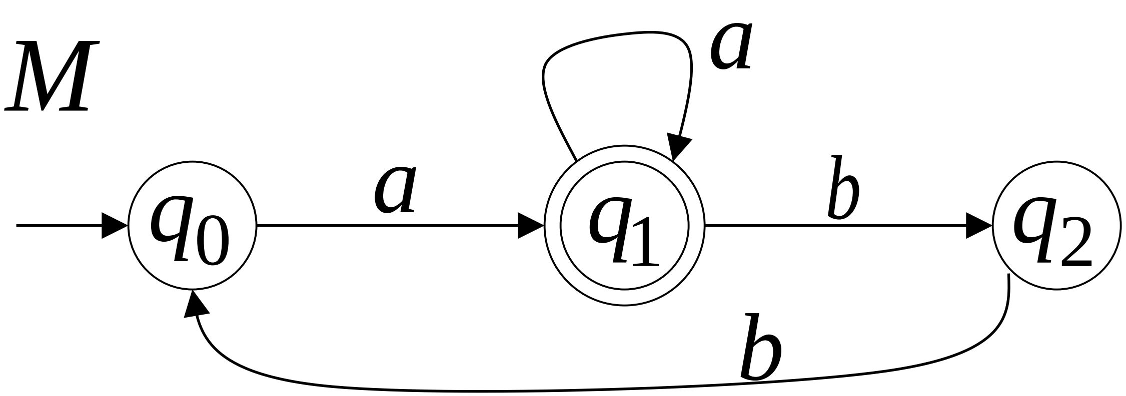


DFA M'

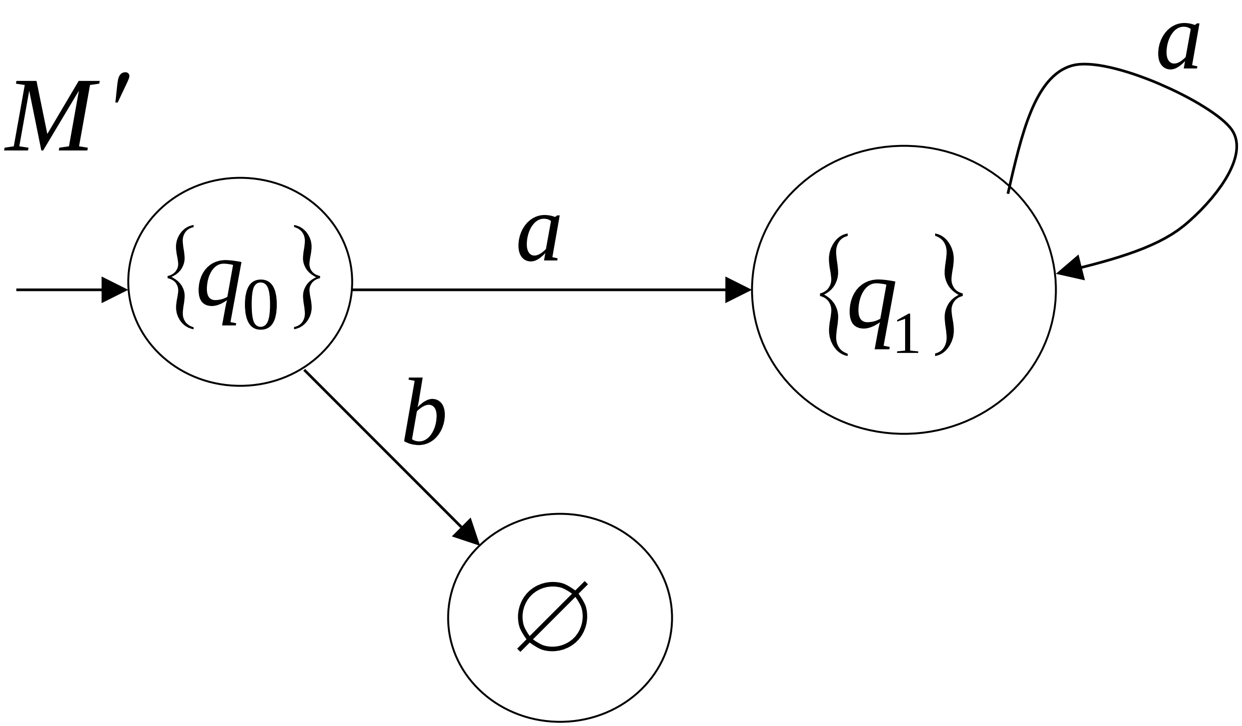


Trap/dead state

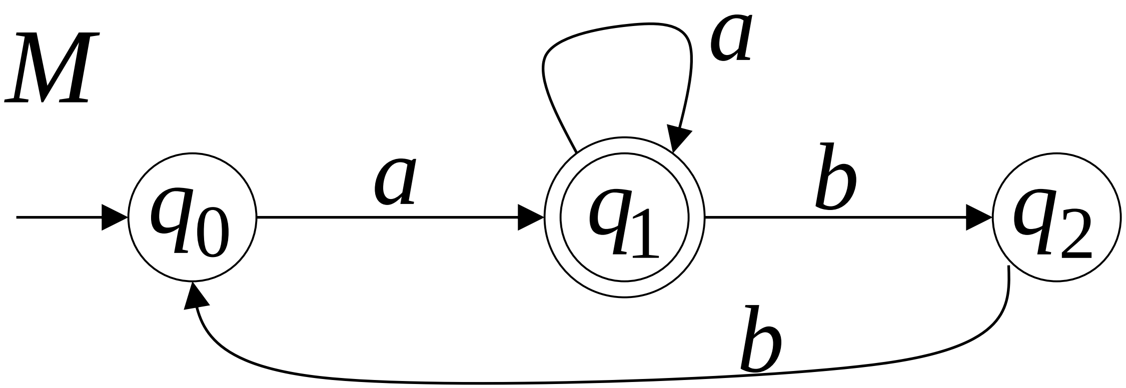
NFA M



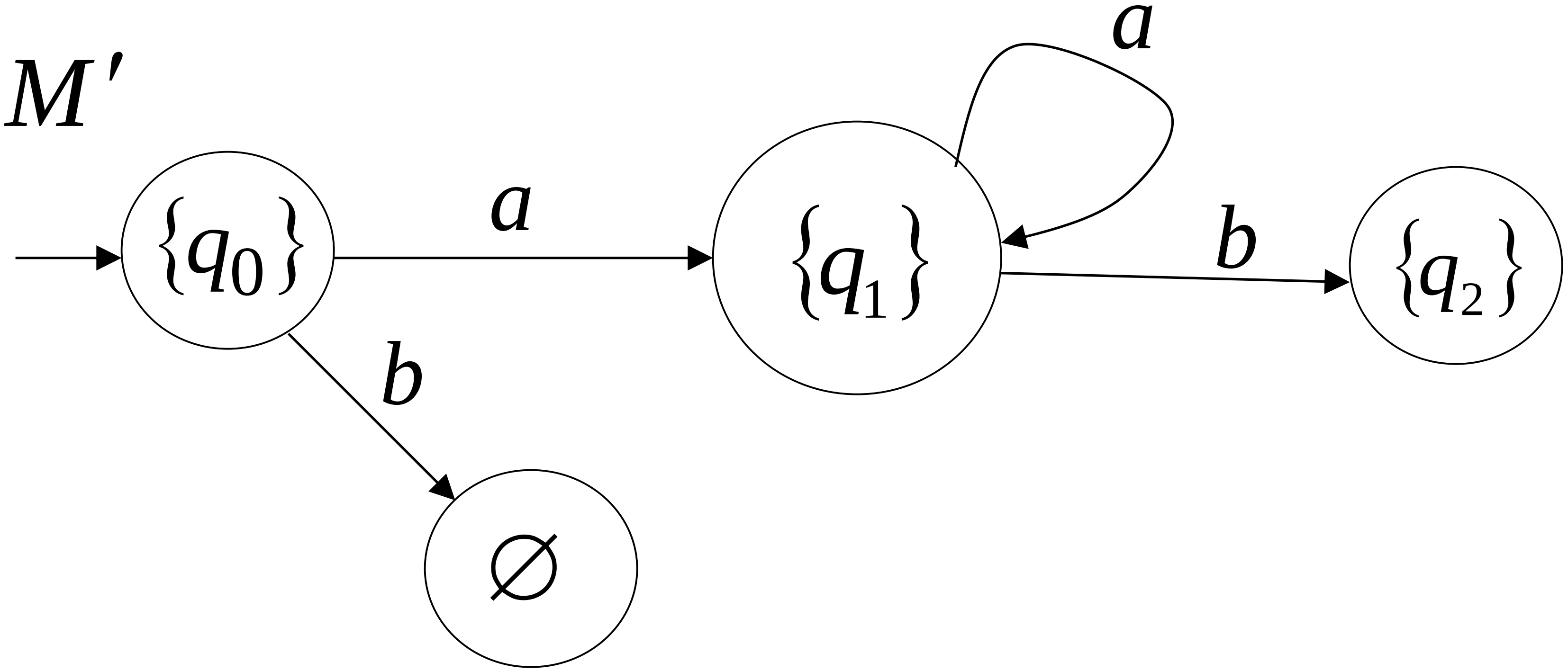
DFA M'



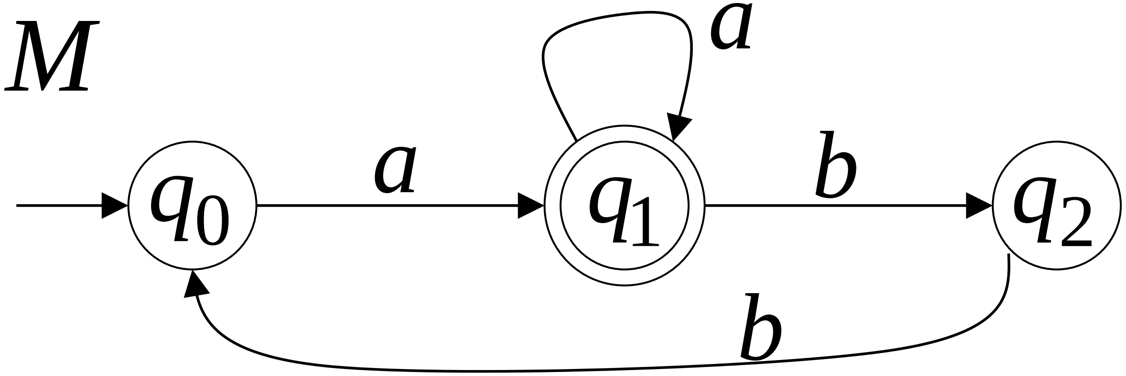
NFA M



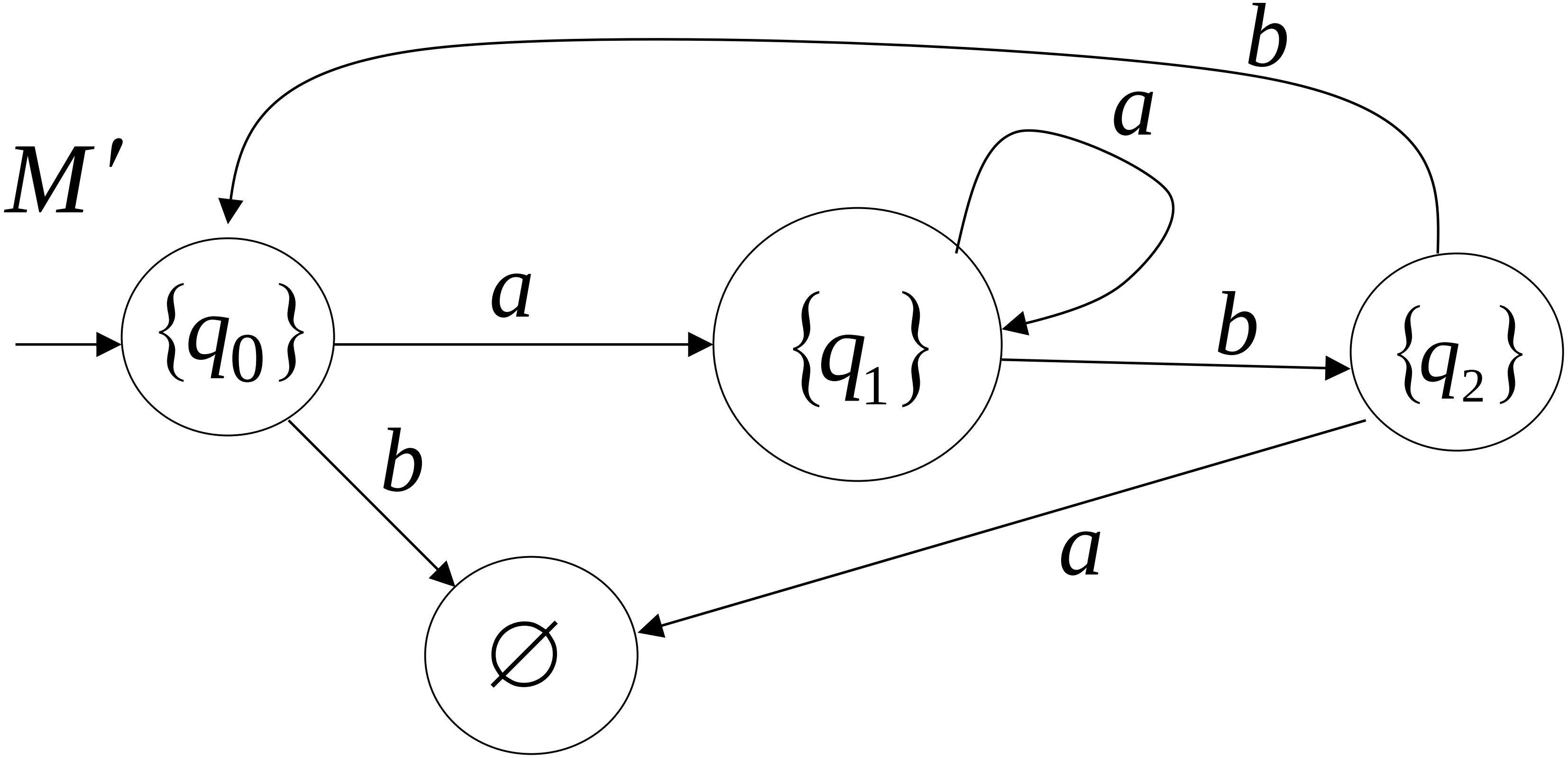
DFA M'



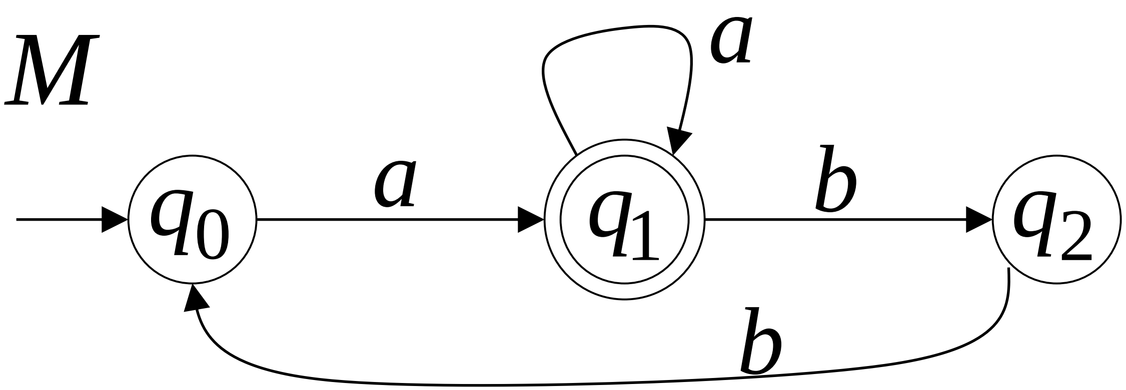
NFA M



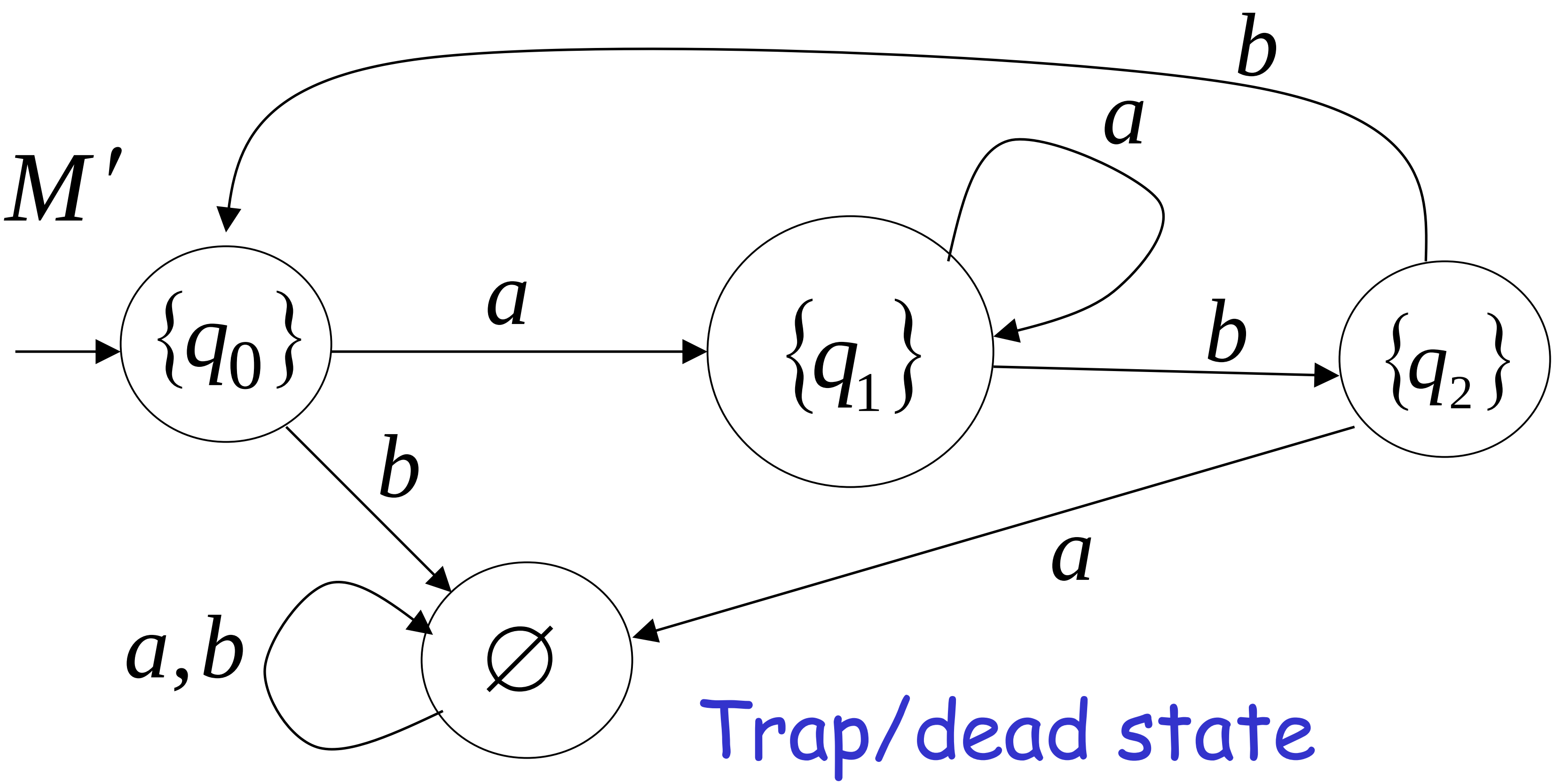
DFA M'



NFA M

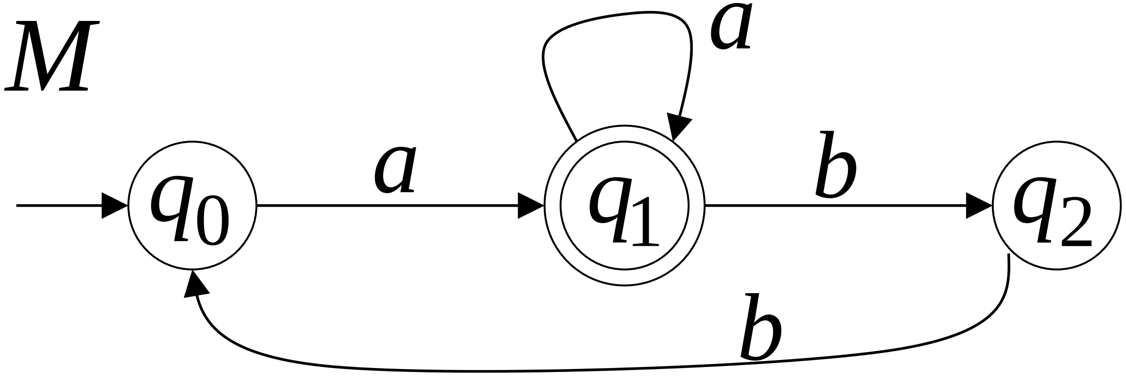


DFA M'



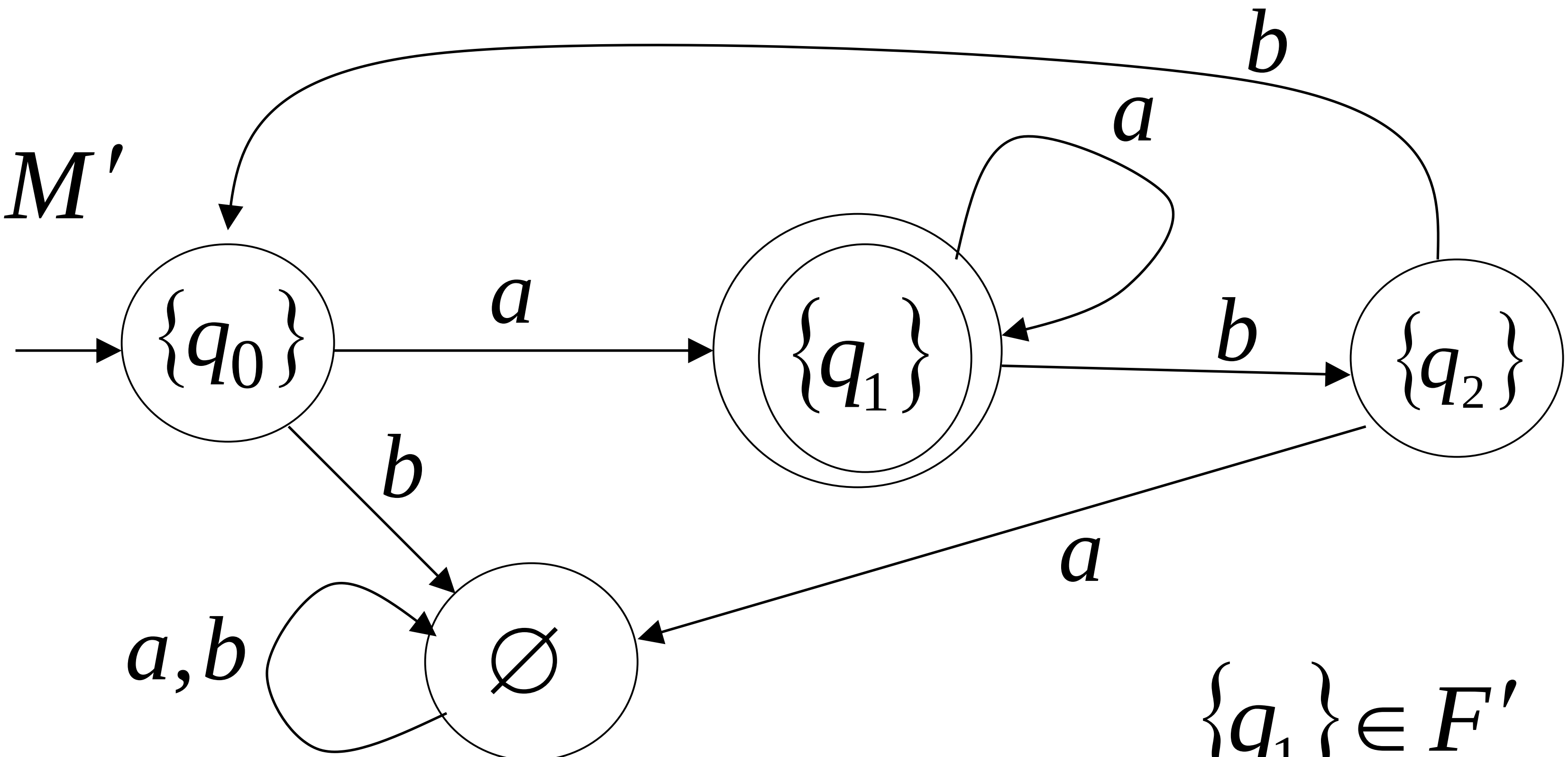
END OF CONSTRUCTION

NFA M



$q_1 \in F$

DFA M'



$\{q_1\} \in F'$

General Conversion Procedure

Input: an NFA M

Output: an equivalent DFA M'

with $L(M) = L(M')$

The NFA has states $q_0, q_1, q_2, \ldots$

The DFA has states from the power set

$$\emptyset, \{q_0\}, \{q_1\}, \{q_0, q_1\}, \{q_1, q_2, q_3\}, \ldots$$

Conversion Procedure Steps

step

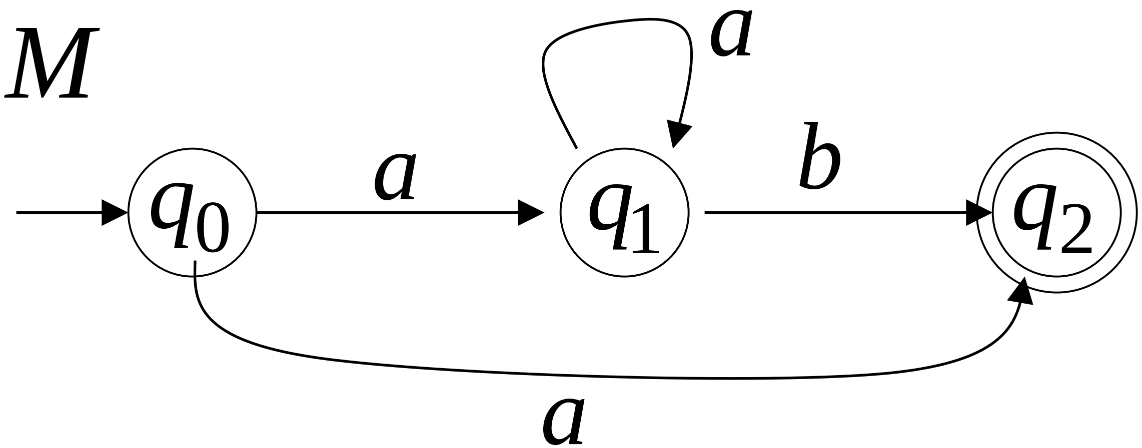
1. Initial state of NFA: q_0

$$\Downarrow$$

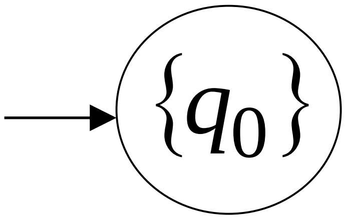
Initial state of DFA: $\{q_0\}$

Example

NFA M

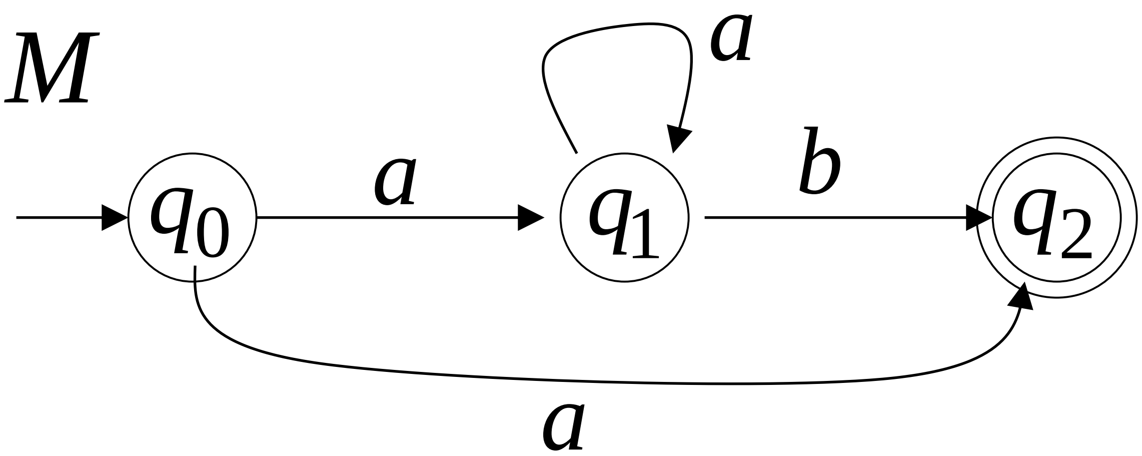


DFA M'



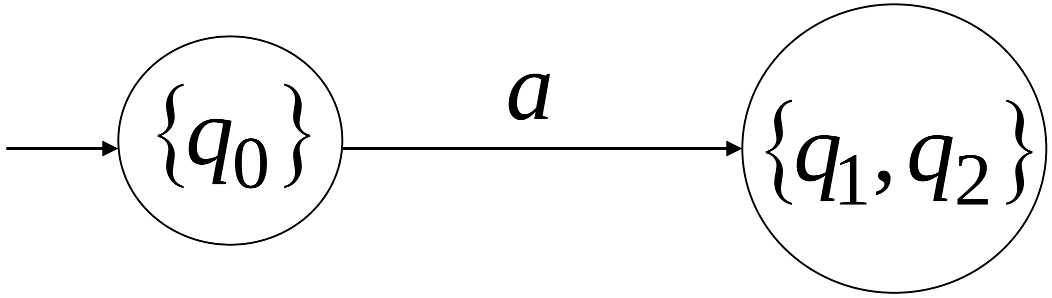
Example $\delta^*(q_0, a) = \{q_1, q_2\}$

NFA M



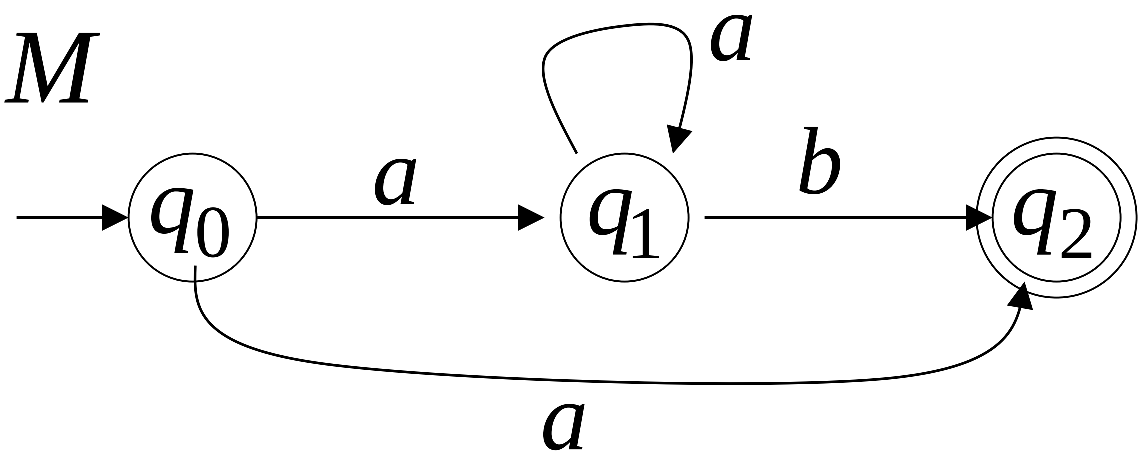
$\delta(\{q_0\}, a) = \{q_1, q_2\}$

DFA M'

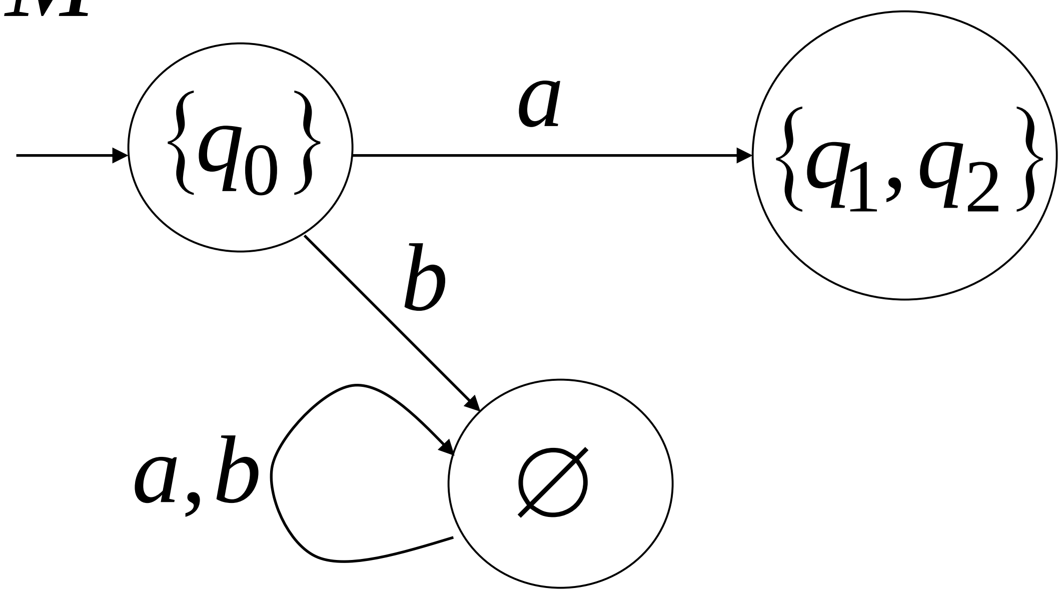


Example

NFA M



DFA M'



step

2. For every DFA's state $\{q_i, q_j, \ldots, q_m\}$

compute in the NFA

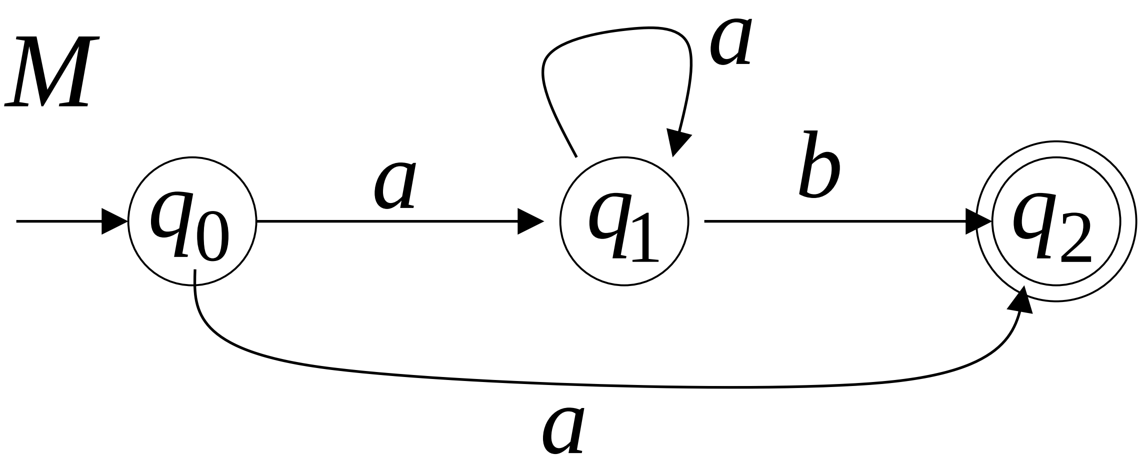
$$\left.\begin{array}{r} \delta^*(q_i, a) \\ \cup\, \delta^*(q_j, a) \\ \ldots \\ \cup\, \delta^*(q_m, a) \end{array}\right\} \overset{\text{Union}}{=} \{q'_k, q'_l, \ldots, q'_n\}$$

add transition to DFA

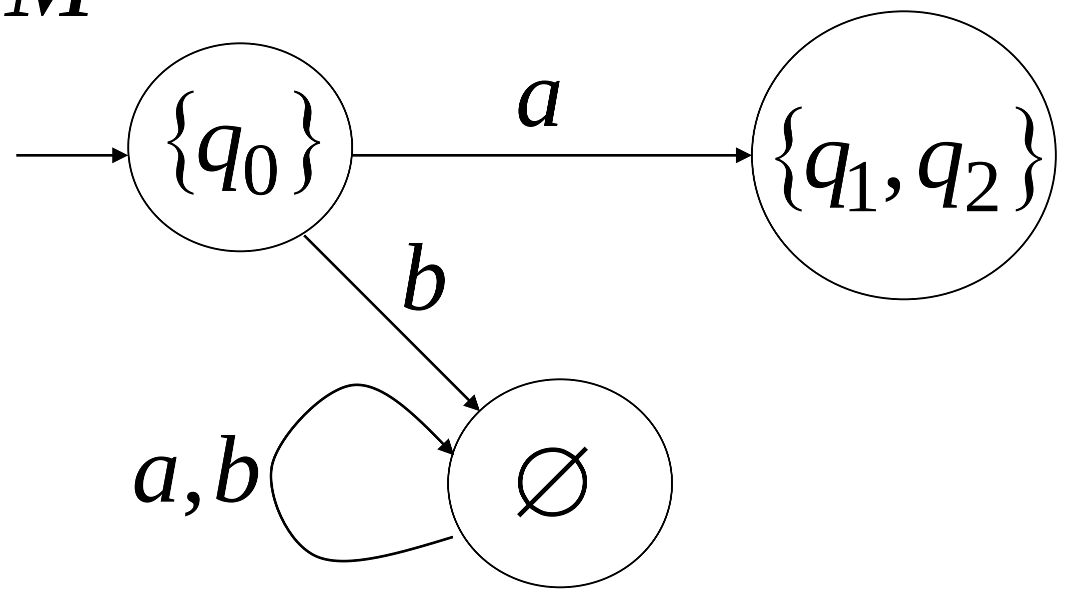
$$\delta(\{q_i, q_j, \ldots, q_m\},\; a) = \{q'_k, q'_l, \ldots, q'_n\}$$

Example

NFA M

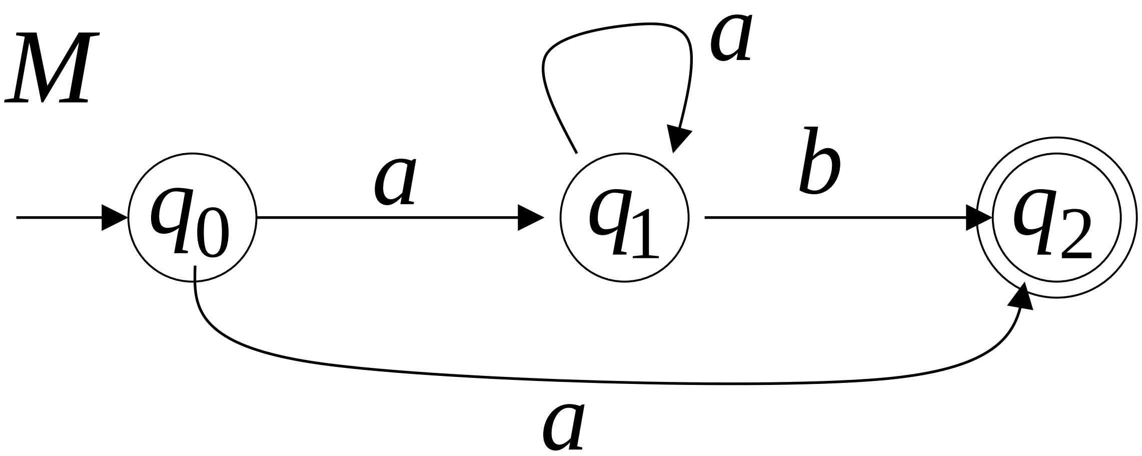


DFA M'

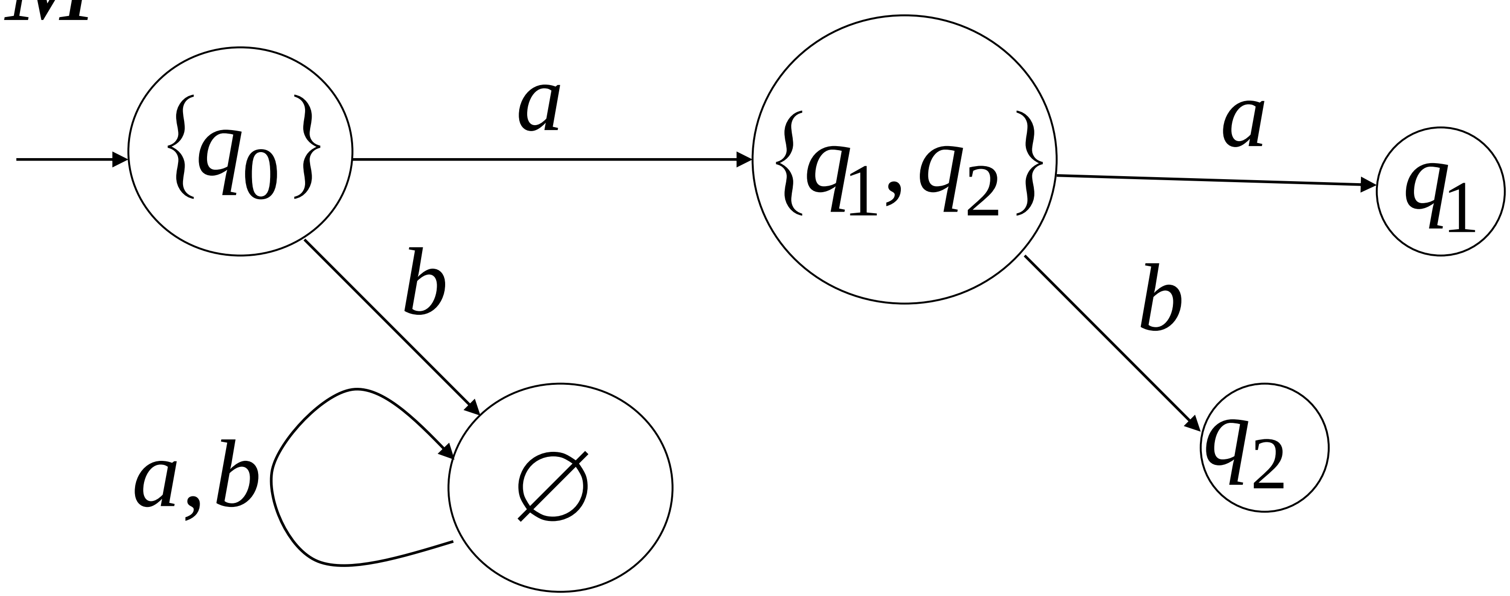


Example

NFA M



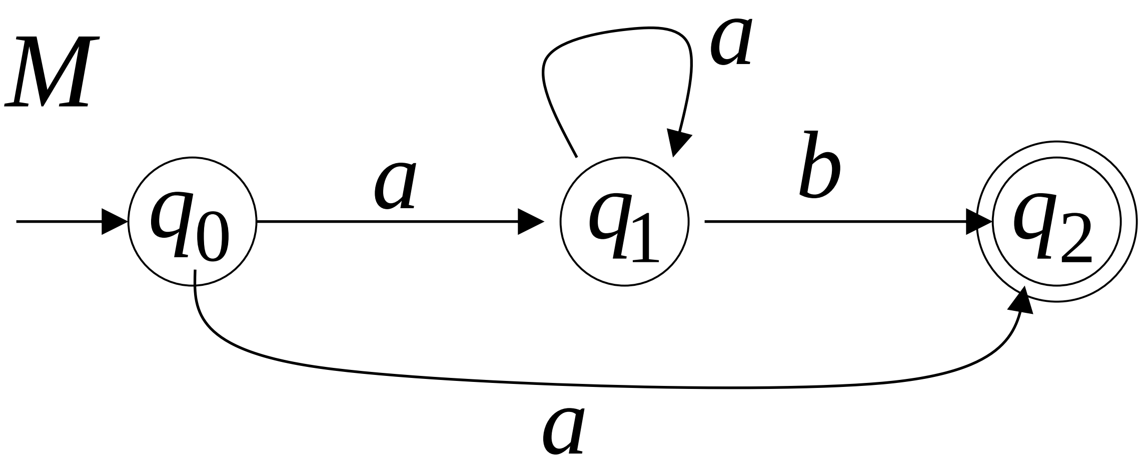
DFA M'



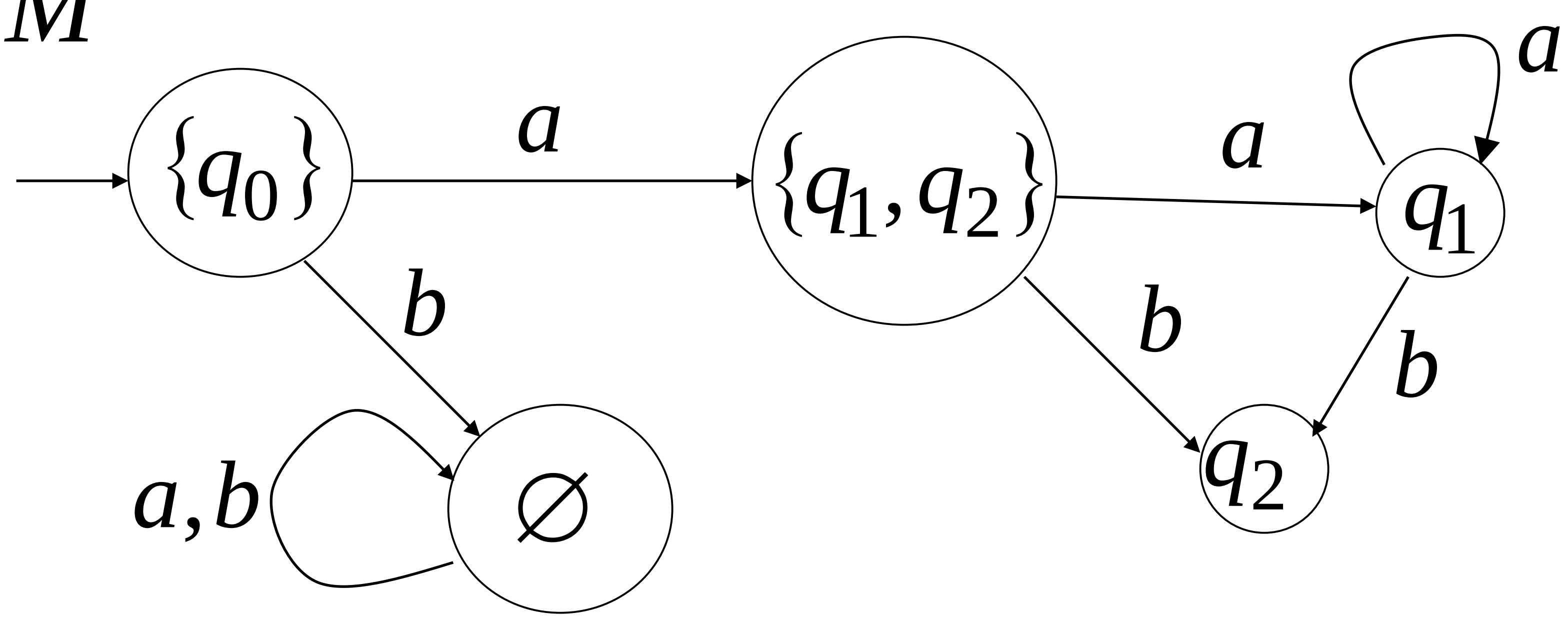
step 3. Repeat Step 2 for every state in DFA and symbols in alphabet until no more states can be added in the DFA

Example

NFA M

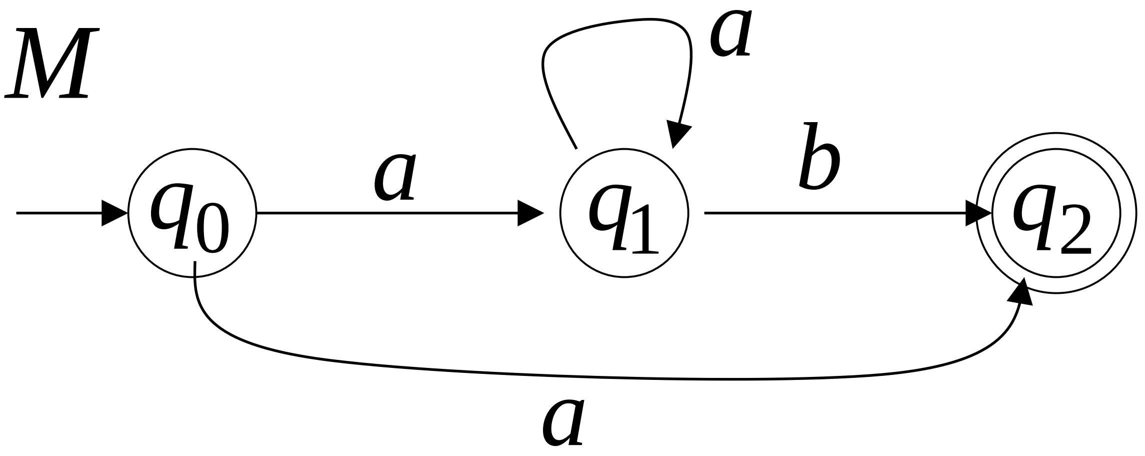


DFA M'

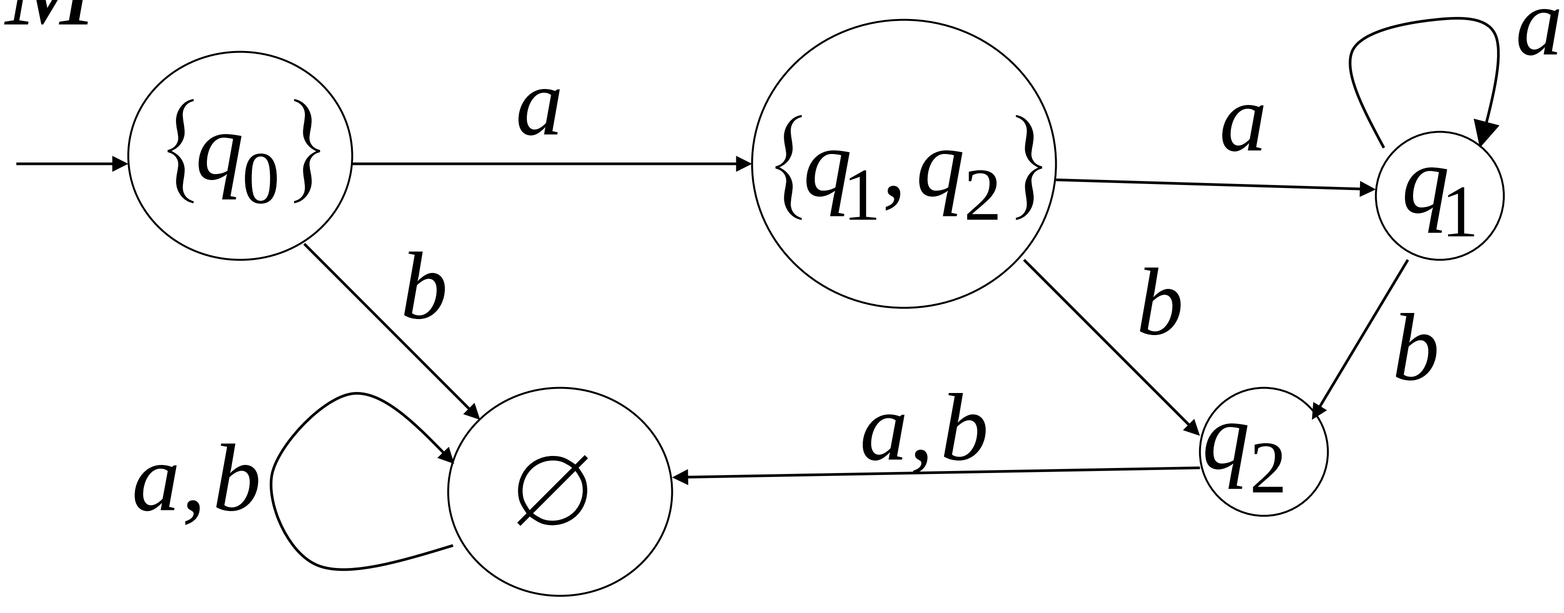


Example

NFA M



DFA M'



step

4. For any DFA state $\{q_i, q_j, \ldots, q_m\}$

if some q_j is accepting state in NFA

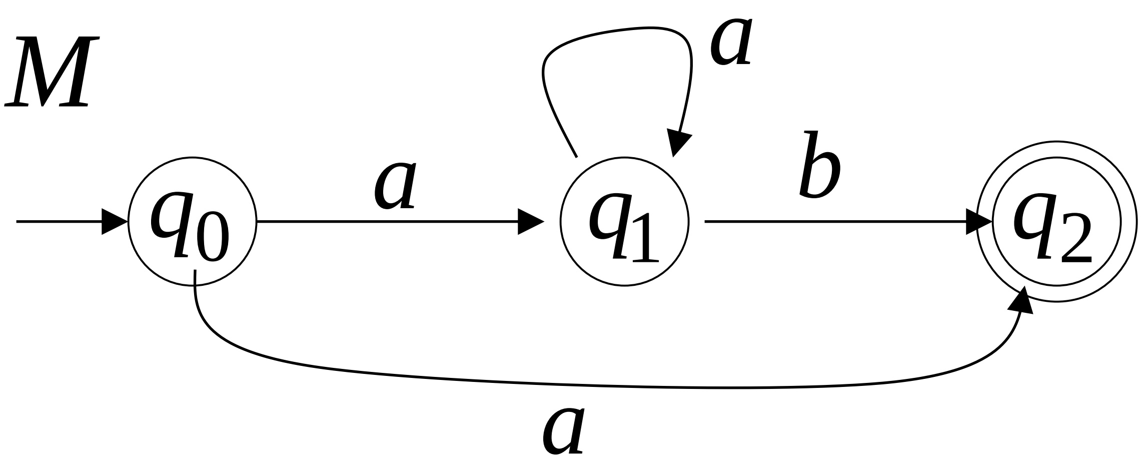
Then, $\{q_i, q_j, \ldots, q_m\}$

is accepting state in DFA

Example

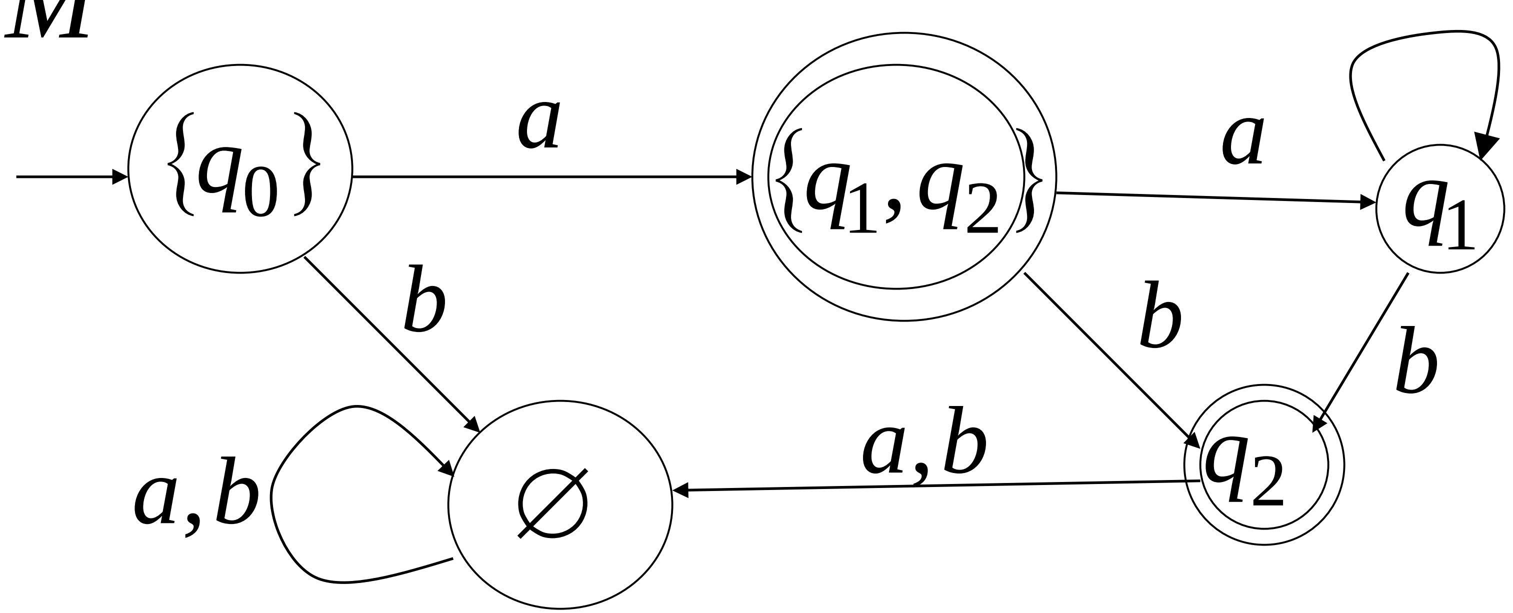
NFA M

$q_2 \in F$



$\{q_1, q_2\} \in F'$

DFA M'



Review

- NFA and NFA with null(or epsilon) transition
- Minimization of DFA
- NFA to DFA Conversion using subset construction

Lemma:

If we convert NFA M to DFA M'

then the two automata are equivalent:

$$L(M) = L(M')$$

NFA $\Longrightarrow$ DFA

DFA $\Longrightarrow$ NFA

NFA $\Longleftrightarrow$ DFA

- Algorithm
- **Input** – An NDFA
- **Output** – An equivalent DFA
- **Step 1** – Create state table from the given NDFA.
- **Step 2** – Create a blank state table under possible input alphabets for the equivalent DFA.
- **Step 3** – Mark the start state of the DFA by q0 (Same as the NDFA).
- **Step 4** – Find out the combination of States $\{Q_0, Q_1, \ldots, Q_n\}$ for each possible input alphabet.
- **Step 5** – Each time we generate a new DFA state under the input alphabet columns, we have to apply step 4 again, otherwise go to step 6.
- **Step 6** – The states which contain any of the final states of the NDFA are the final states of the equivalent DFA.

NFA

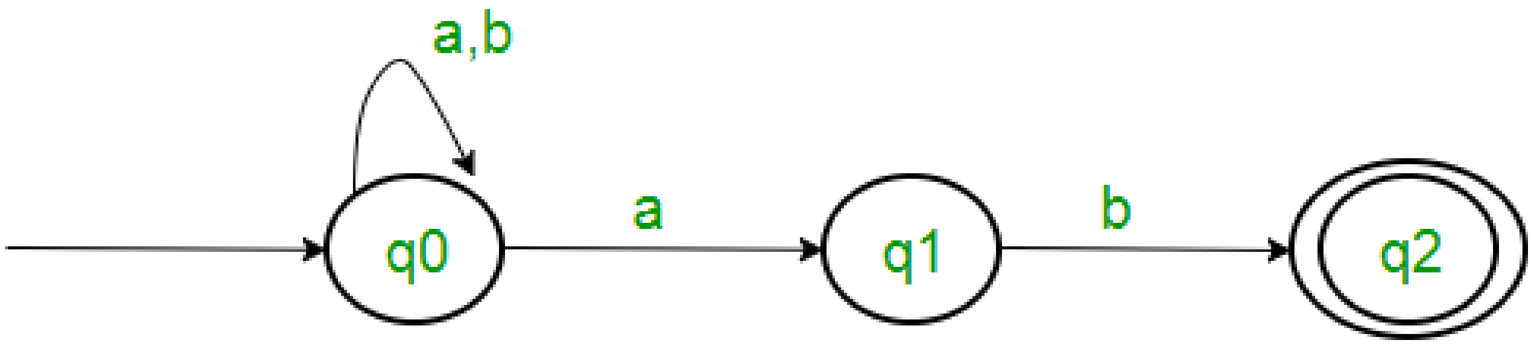


Figure 1

$Q=\{q0,q1,q2\}$

$\Sigma=a,b$

$q_0=q0$

$F=\{q2\}$

δ (Transition Function)

	a	b
q0	{q0,q1}	q0
q1		q2
q2		

	a	b
q0	{q0,q1}	q0
q1		q2
q2		

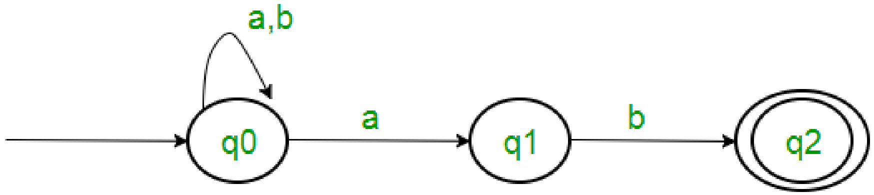
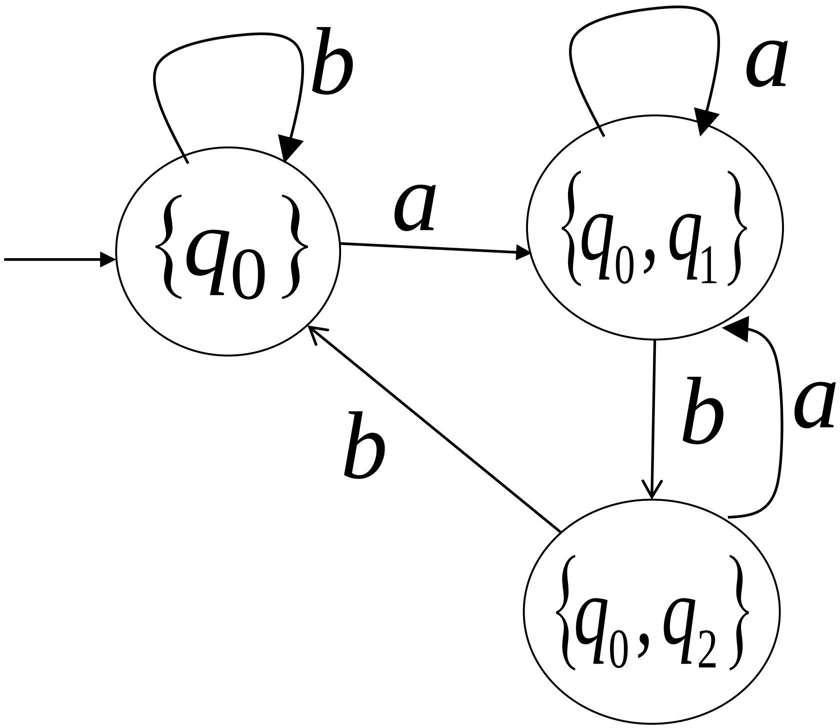
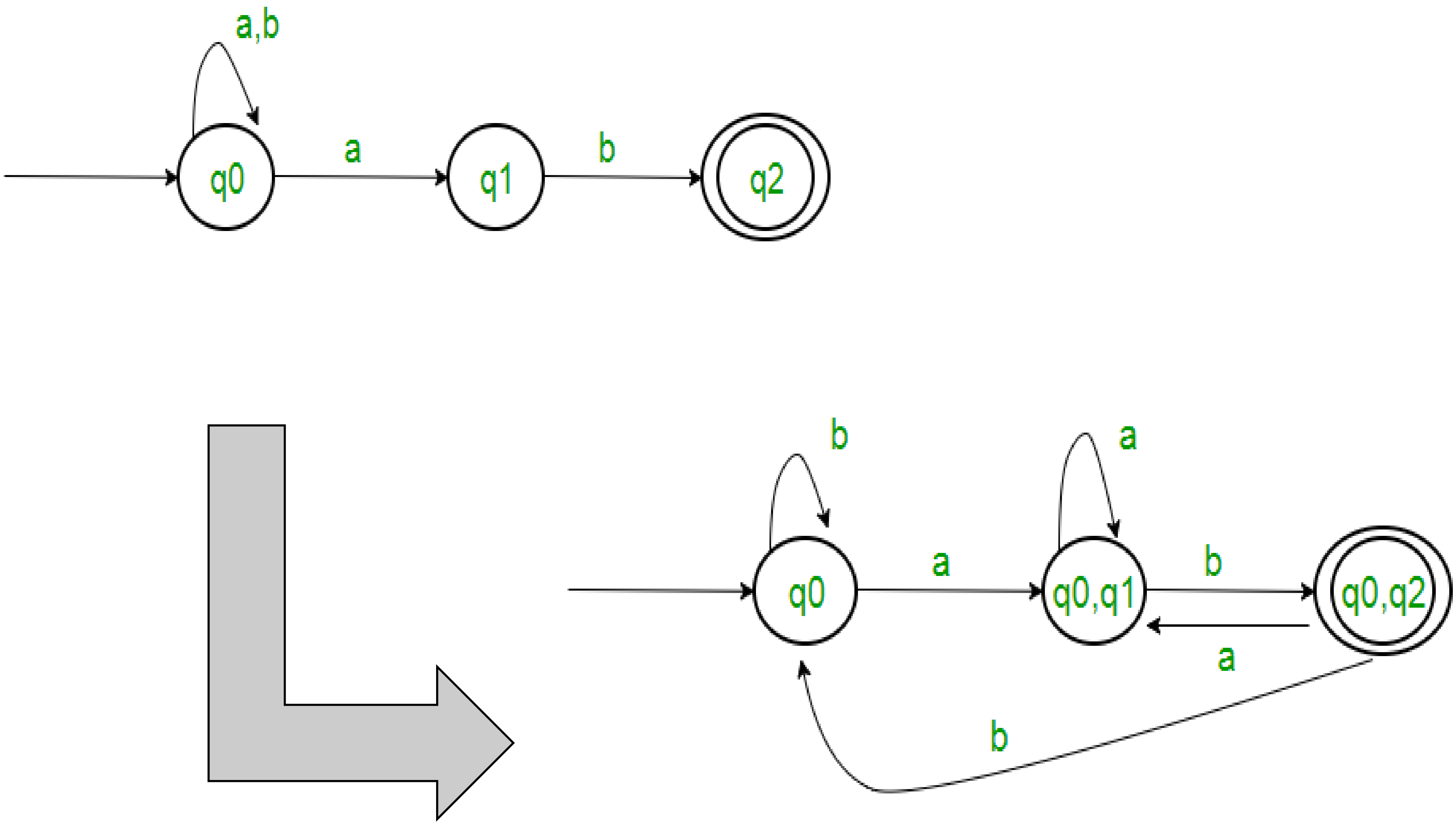


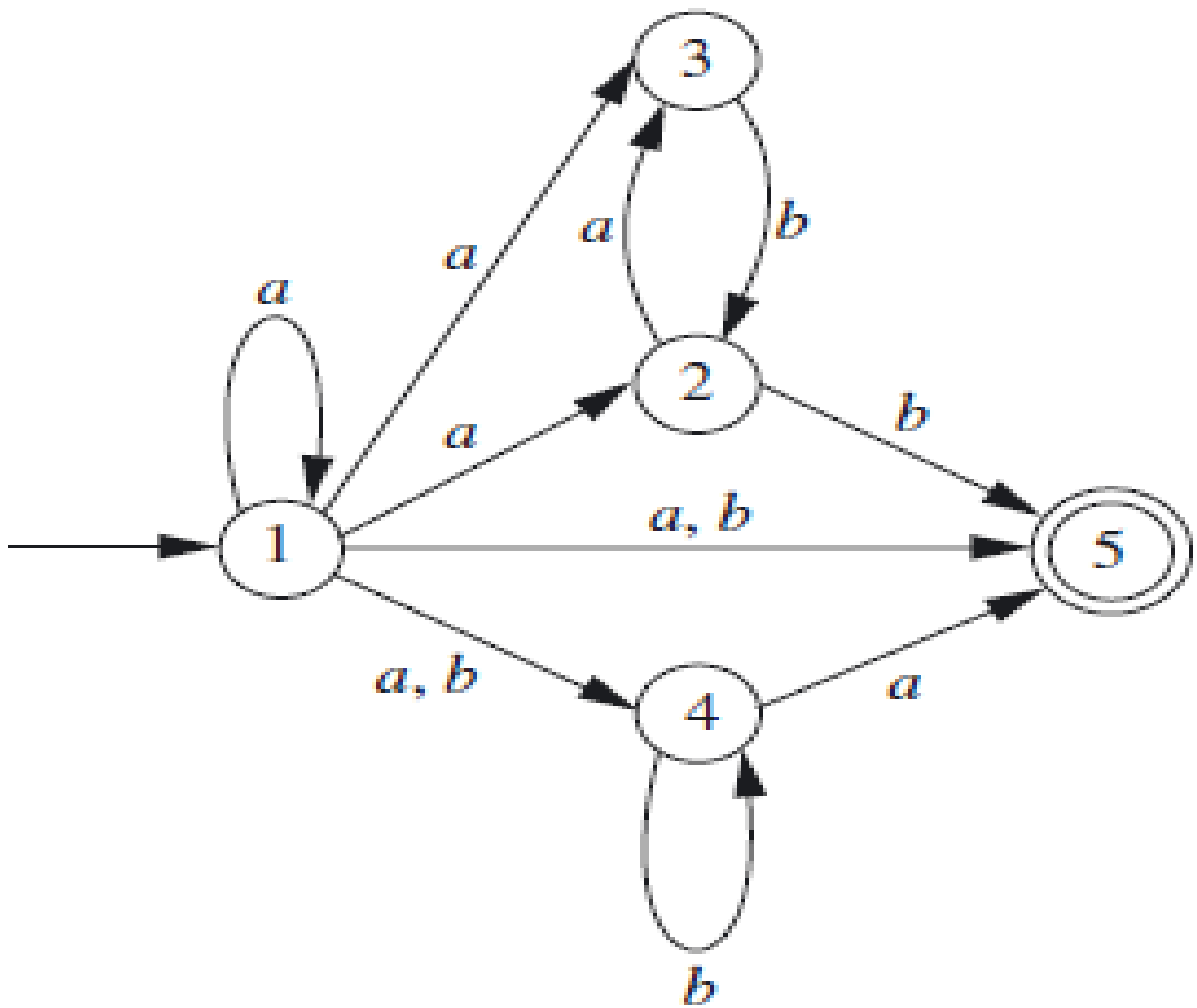
Figure 1

State	a	B
q0	{q0,q1}	q0
{q0,q1}	{q0,q1}	{q0,q2}
{q0,q2}	{q0,q1}	q0

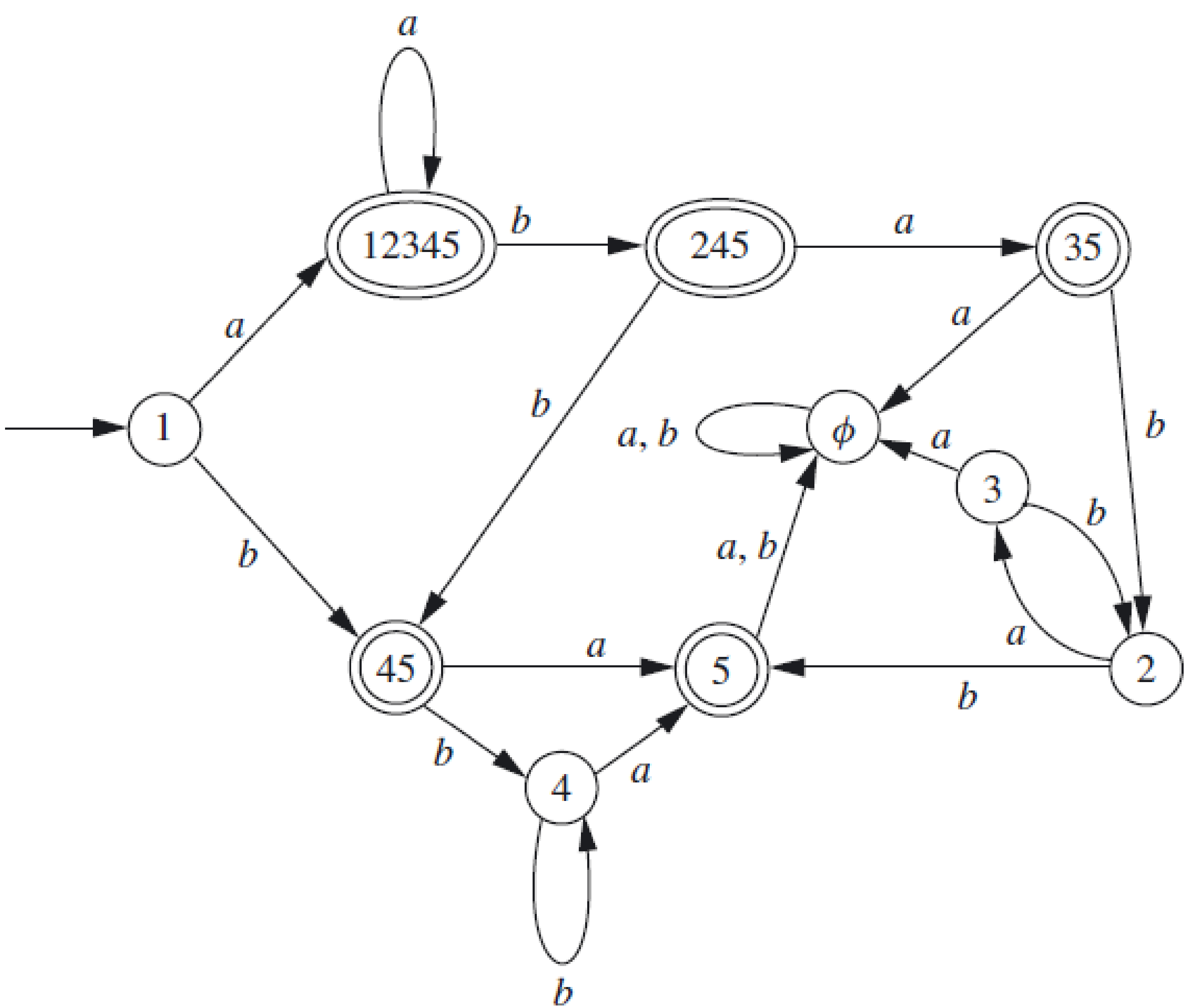



a,b
a
b
q0
q1
q2
b
a
a
b
a
b
q0
q0,q1
q0,q2

Example for NFA to DFA Conversion

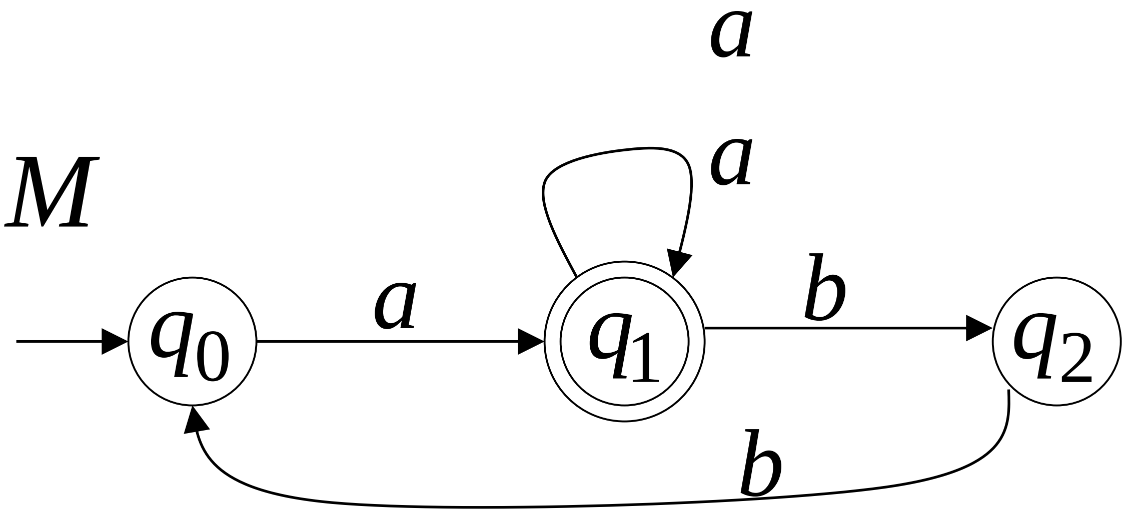


Output

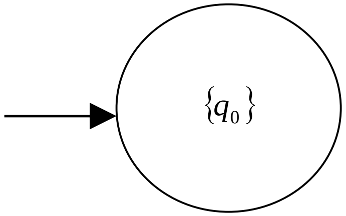


Conversion NFA-λ to NFA

NFA M



DFA M'



Review

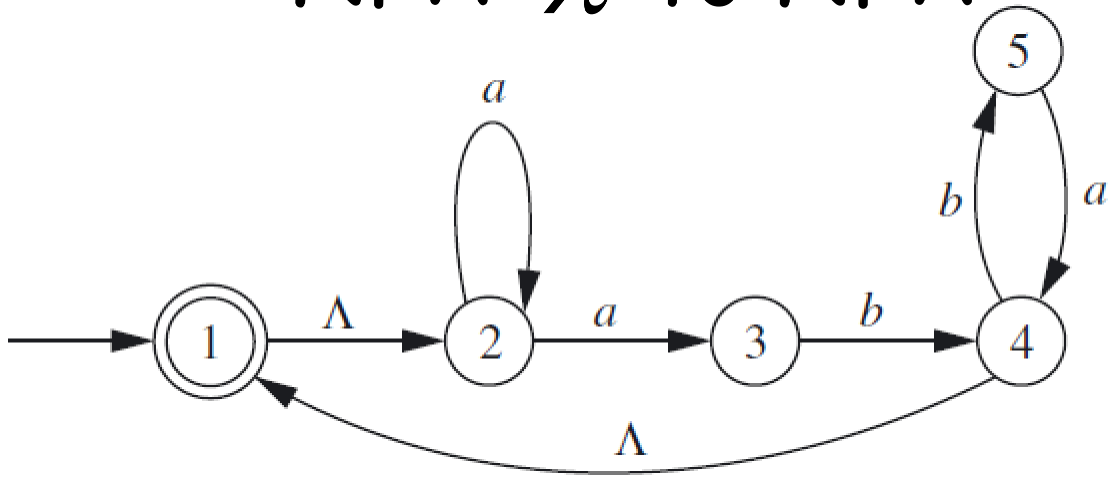
- Minimization of DFA
- NFA to DFA Conversion using subset construction
- NFA- λ to NFA Conversion using subset construction

The λ-Closure of a Set of States

- Suppose *$M = (Q, \lambda, q_0, A, \delta)$ is an NFA, and $S \subseteq Q$ is a set of states.*
- The λ-closure of *S is the set (S) that can be defined recursively as*
- follows.

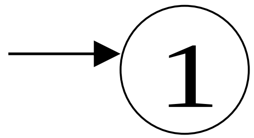
  1. $S \subseteq (S)$.
  2. For every $q \in (S)$, $\delta(q,\lambda) \subseteq (S)$.

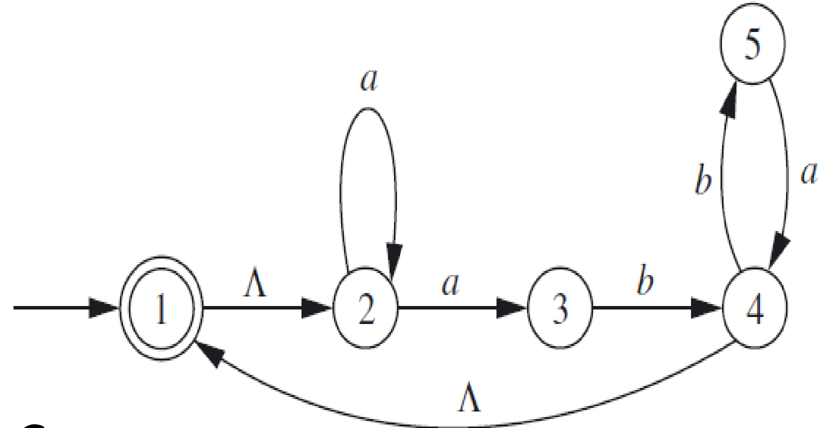
NFA-λ to NFA



1. States of NFA-λ and NFA are Same
2. *Initial state of NFA-λ is initial state of NFA.*
3. *For transitions,*
- *find the null closure of the state*
- *apply the symbol from the alphabet*
- *find the null closure of a set states generated from the above step*

4. Repeat the step No.4 for each state





1. States of NFA- λ and NFA are Same
2. *Initial state of NFA-λ is initial state of NFA.*
3. *For transitions,*

- *find the null closure of the state*
- *apply the symbol from the alphabet*
- *find the null closure of the states generated from the above step*

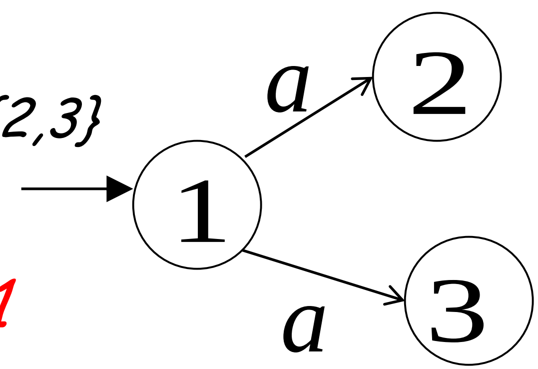
1) λ -({1})= {1,2}

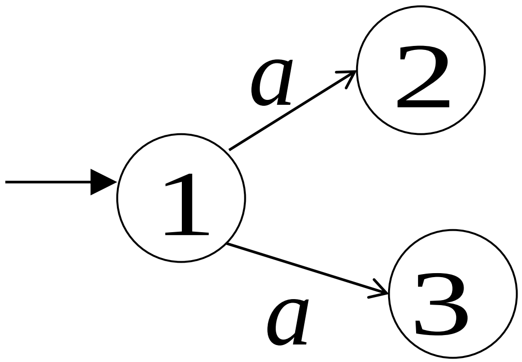
2) $\delta(\{1,2\}, a) = \delta(1, a) U \delta(2, a) =$ Φ U {2,3}={2,3}

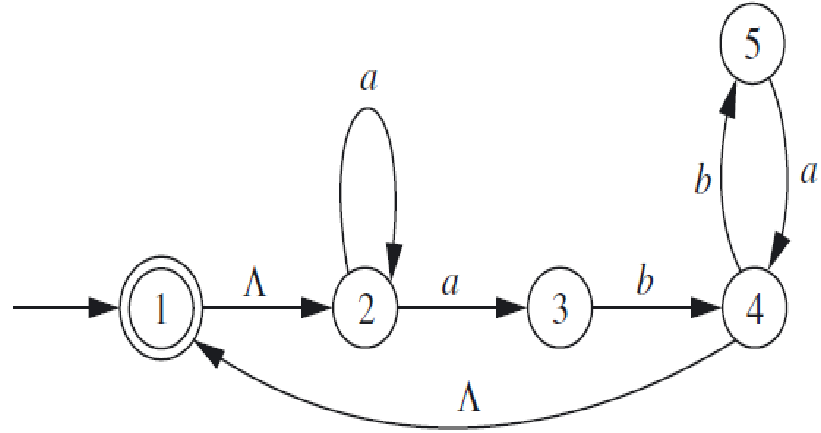
3) λ- {2,3}={2,3}

4) After applying transitions a on state 1

It will move to state {2,3}







1) λ -({1})= {1,2}

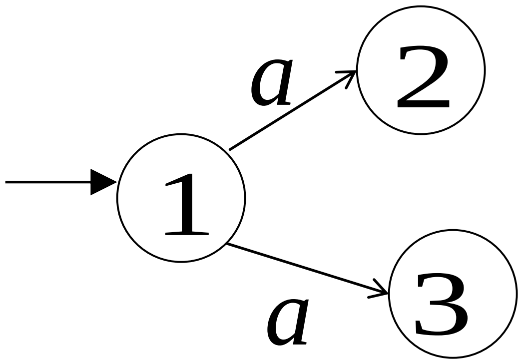
2) $\delta(\{1,2\}, a) = \delta(1, a) \cup \delta(2, a) = \Phi \cup \{2,3\}$

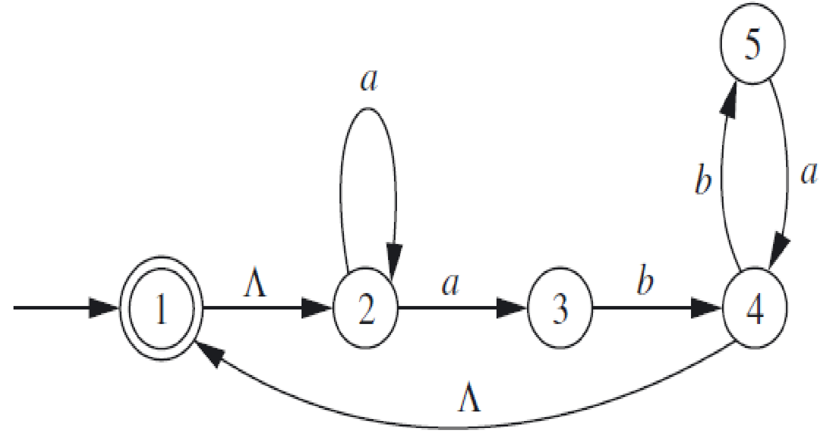
3) λ - {2,3}={2,3}

4) After applying transitions a on state 1

It will move to state {2,3}

		a			b	
	λ		λ	λ		λ
1	{1,2}	{2,3}	{2,3}			





1) λ -({1})= {1,2}

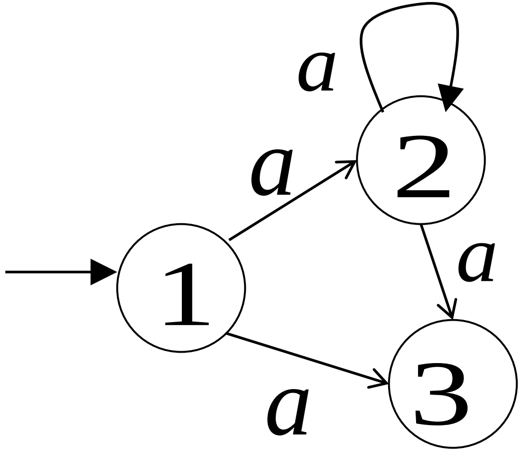
2) $\delta(\{1,2\}, b) = \delta(1, b) U \delta(2, b) =$ Φ U Φ

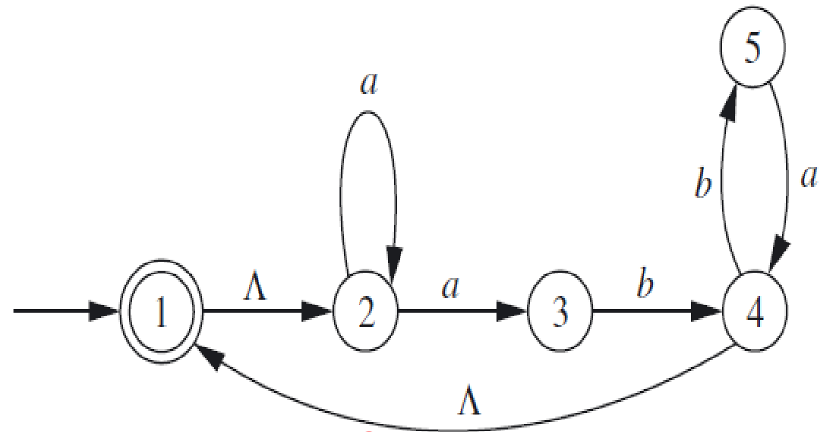
3) λ - Φ = Φ

4) After applying transitions b on state 1

It will move to the dead state

		a			b	
	λ		λ	λ		λ
1	{1,2}	{2,3}	{2,3}	{1,2}	Φ	Φ





1) λ -({2})= {2}

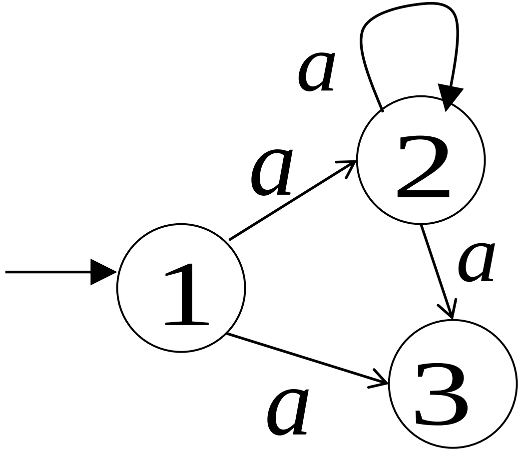
2) $\delta(\{2\}, a) =$ {2,3}

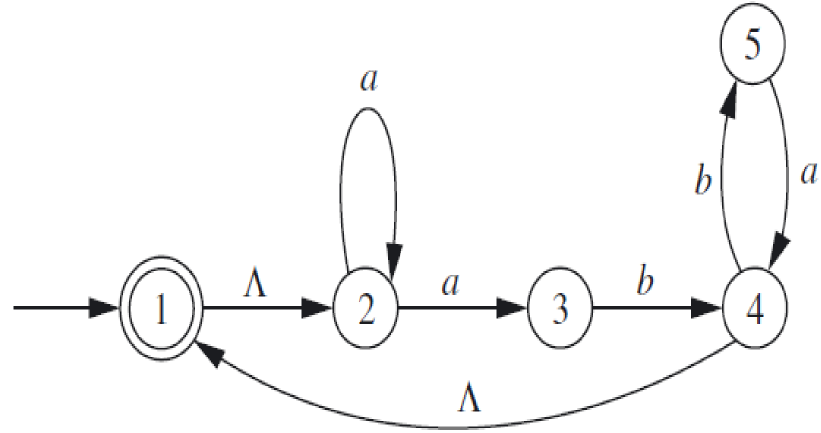
3) λ - {2,3}={2,3}

4) After applying transitions a on state 2

It will move to state {2,3}

		a			b	
	λ		λ	λ		λ
1	{1,2}	{2,3}	{2,3}	{1,2}	Φ	Φ
2	{2}	{2,3}	{2,3}			





1) λ -({2})= {2}

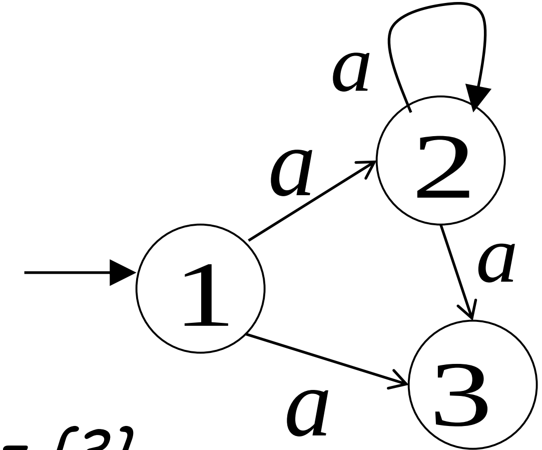
2) $\delta(\{2\}, b) = \Phi$

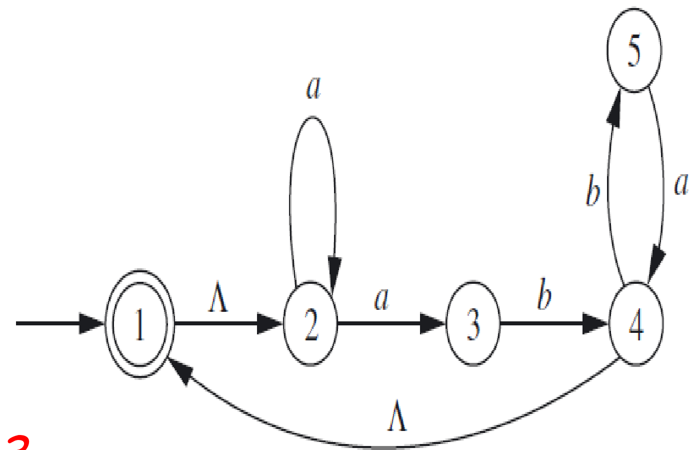
3) λ - Φ=Φ

4) After applying transitions b on state 2

It will move to Dead state.

		a			b	
	λ		λ	λ		λ
1	{1,2}	{2,3}	{2,3}	{1,2}	Φ	Φ
2	{2}	{2,3}	{2,3}	{2}	Φ	Φ





1) λ-({3})= {3}

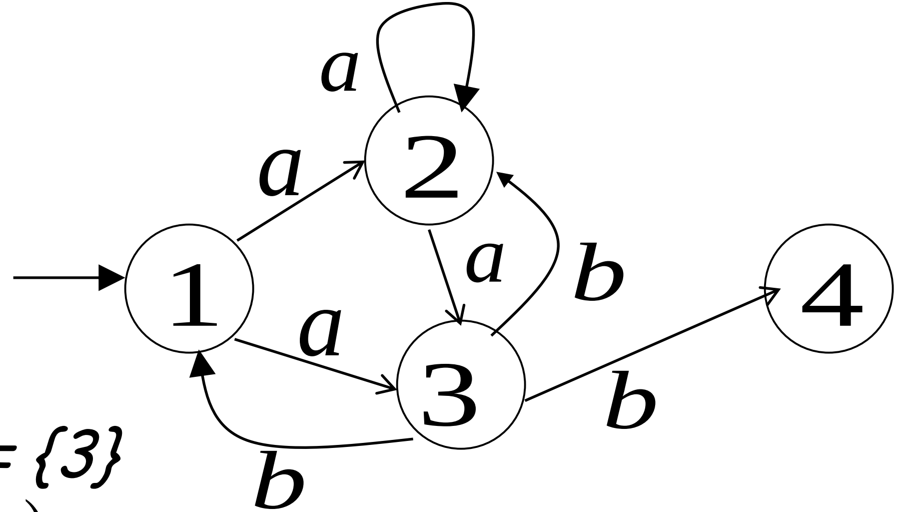
2) $\delta(\{3\}, a) = \Phi$

3) λ-Φ = Φ

4) After applying transitions a on state 3

It will move to Dead State

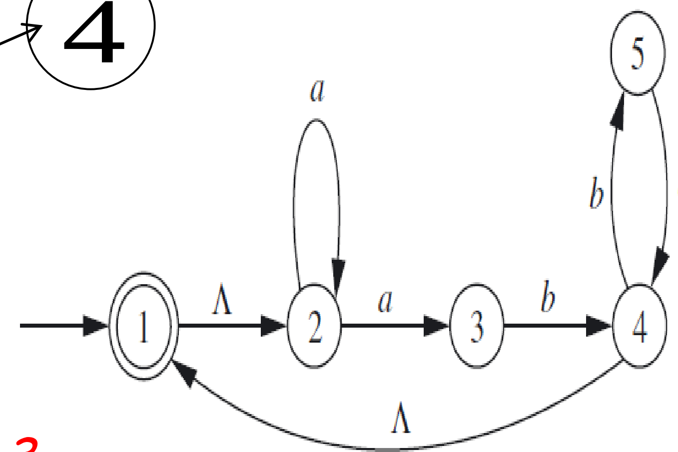
		a			b	
	λ		λ	λ		λ
1	{1,2}	{2,3}	{2,3}	{1,2}	Φ	Φ
2	{2}	{2,3}	{2,3}	{2}	Φ	Φ
3	{3}	Φ	Φ			
4						
5						



1) λ-({3}) = {3}

2) $\delta(\{3\}, b) =$ 4

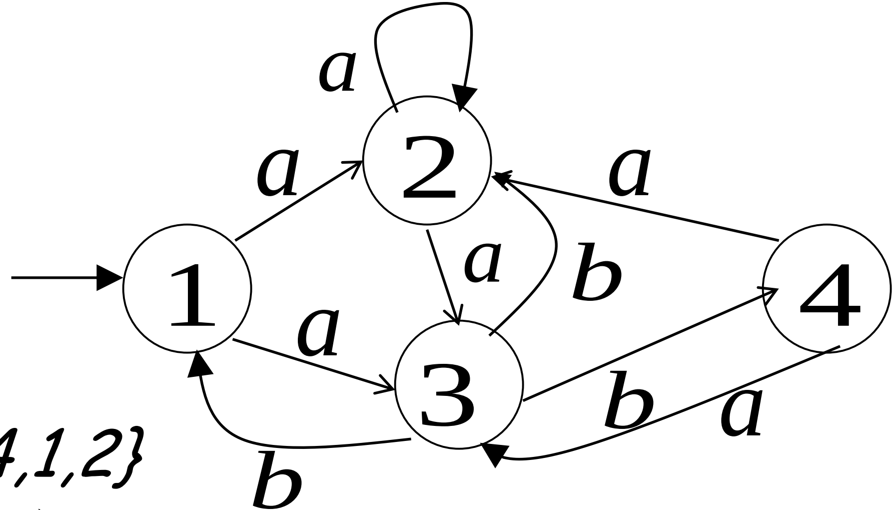
3) λ-{4} = {4,1,2}



4) After applying transitions b on state 3

It will move to State 1, 2 and 4

		a			b	
	λ		λ	λ		λ
1	{1,2}	{2,3}	{2,3}	{1,2}	Φ	Φ
2	{2}	{2,3}	{2,3}	{2}	Φ	Φ
3	{3}	Φ	Φ	{3}	{4}	{4,1,2}
4						
5						



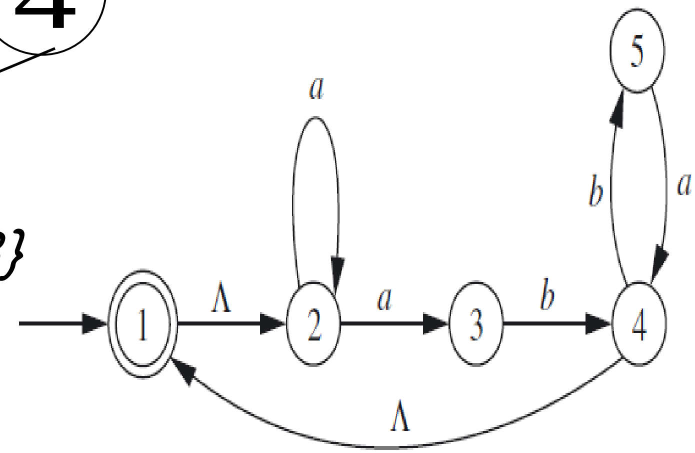
1) λ -({4})= {4,1,2}

2) $\delta(\{4,1,2\}, a) = \delta(4, a) U \delta(1, a) U \delta(2, a)$ ={2,3}

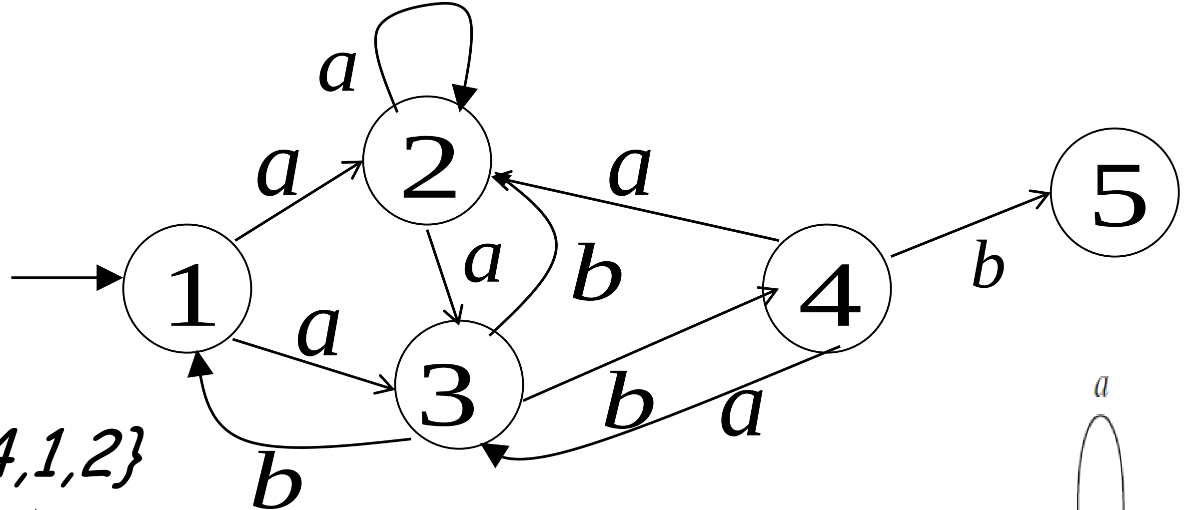
3) λ-{2,3} ={2,3}

4) After applying transitions a on state 4

It will move to State 2 and 3

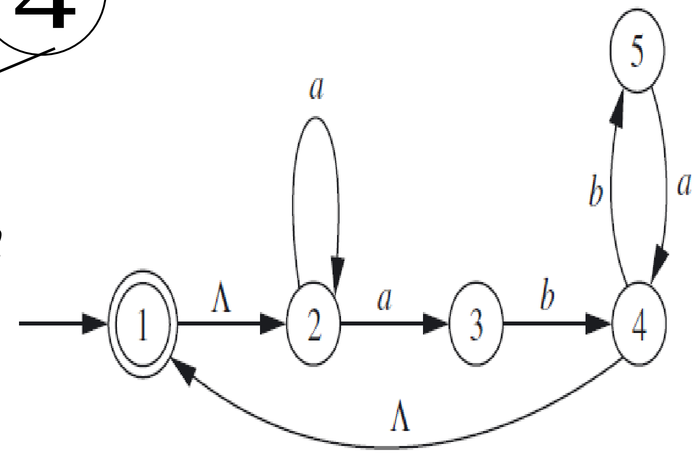


		a			b	
	λ		λ	λ		λ
1	{1,2}	{2,3}	{2,3}	{1,2}	Φ	Φ
2	{2}	{2,3}	{2,3}	{2}	Φ	Φ
3	{3}	Φ	Φ	{3}	{4}	{4,1,2}
4	{4,1,2}	{2,3}	{2,3}			
5						



1) λ -({4})= {4,1,2}

2) $\delta(\{4,1,2\}, b) = \delta(4, b) U \delta(1, b) U \delta(2, b) =$ {5}

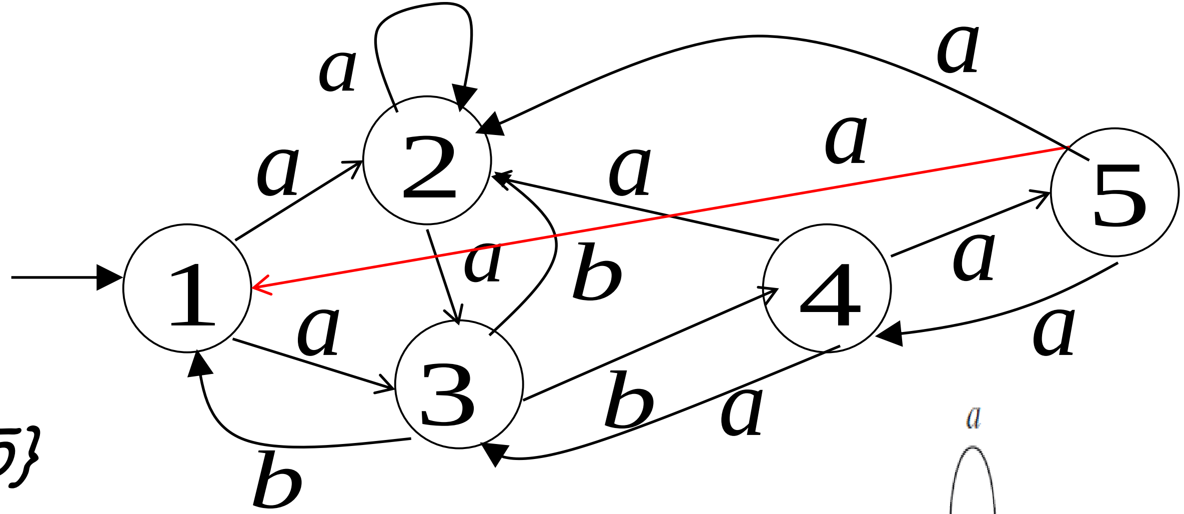


3) λ-{5} ={5}

4) After applying transitions b on state 4

It will move to State 5

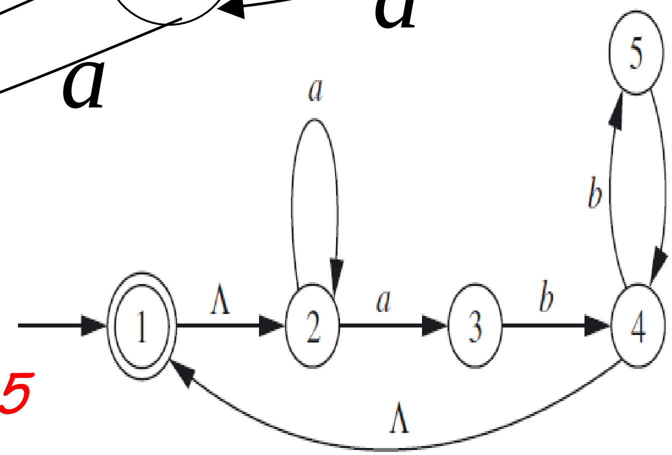
		a			b	
	λ		λ	λ		λ
1	{1,2}	{2,3}	{2,3}	{1,2}	Φ	Φ
2	{2}	{2,3}	{2,3}	{2}	Φ	Φ
3	{3}	Φ	Φ	{3}	{4}	{4,1,2}
4	{4,1,2}	{2,3}	{2,3}	{1,2,4}	{5}	{5}
5						



1) λ -({5})= {5}

2) $\delta(\{5\}, a) =$ {4}

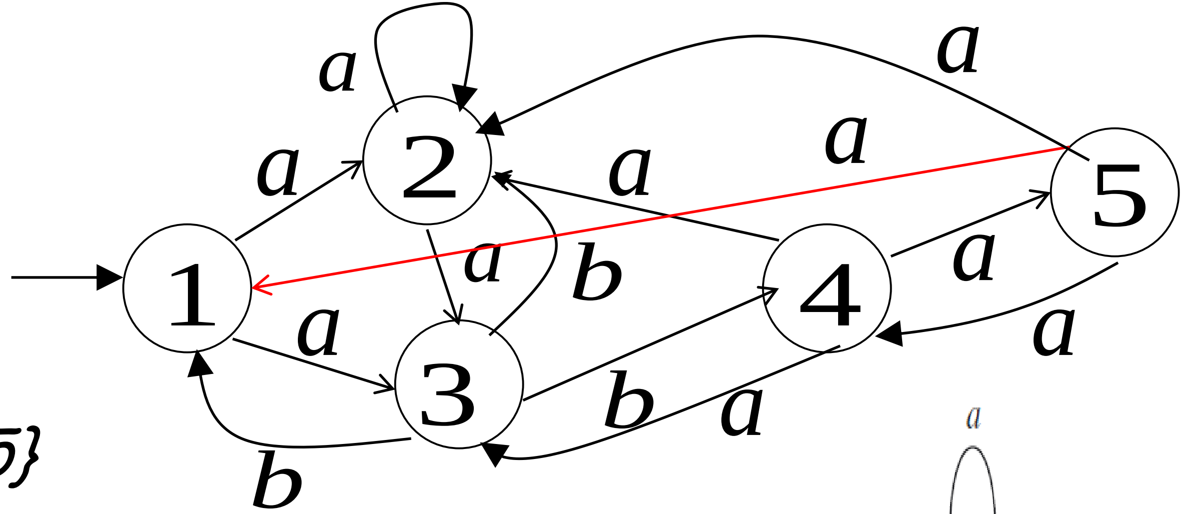
3) λ-{4} ={1,2,4}



4) After applying transitions a on state 5

It will move to State 1,2 and 4

		a			b	
	λ		λ	λ		λ
1	{1,2}	{2,3}	{2,3}	{1,2}	Φ	Φ
2	{2}	{2,3}	{2,3}	{2}	Φ	Φ
3	{3}	Φ	Φ	{3}	{4}	{4,1,2}
4	{4,1,2}	{2,3}	{2,3}	{1,2,4}	{5}	{5}
5	{5}	{4}	{1,2,4}			



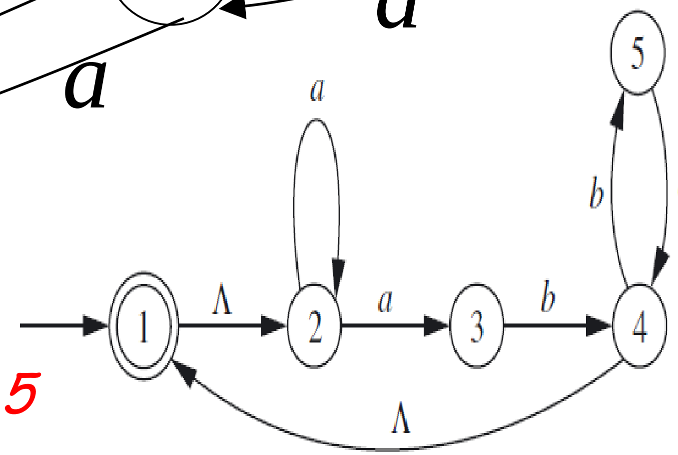
1) λ -({5})= {5}

2) $\delta(\{5\}, b) =$ {Φ}

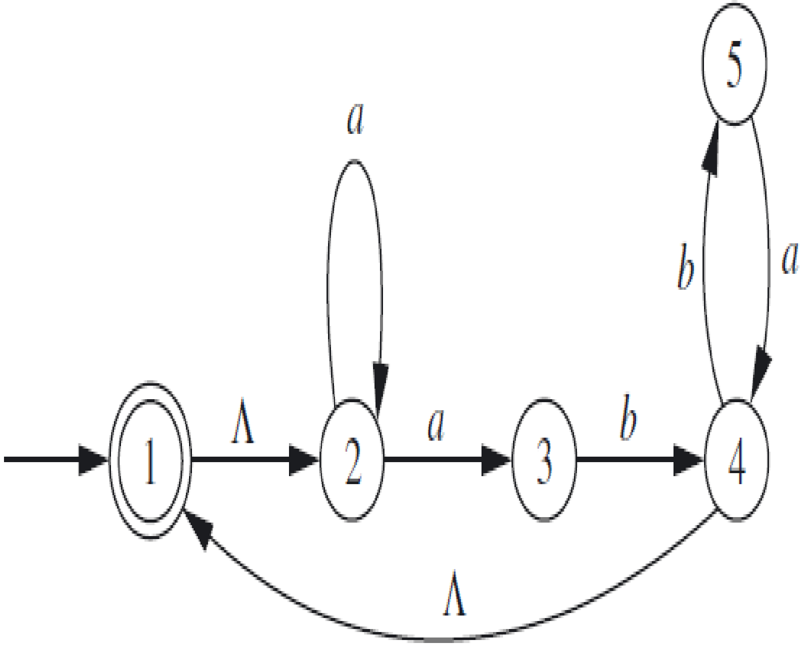
3) λ-{Φ} ={Φ}

4) After applying transitions b on state 5

It will move to Dead State

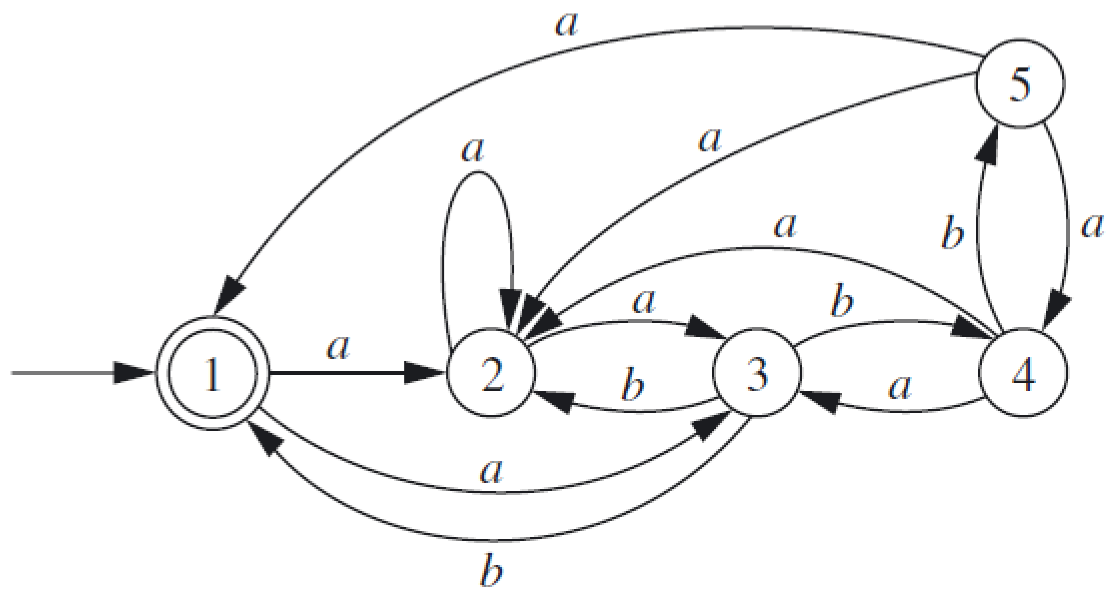


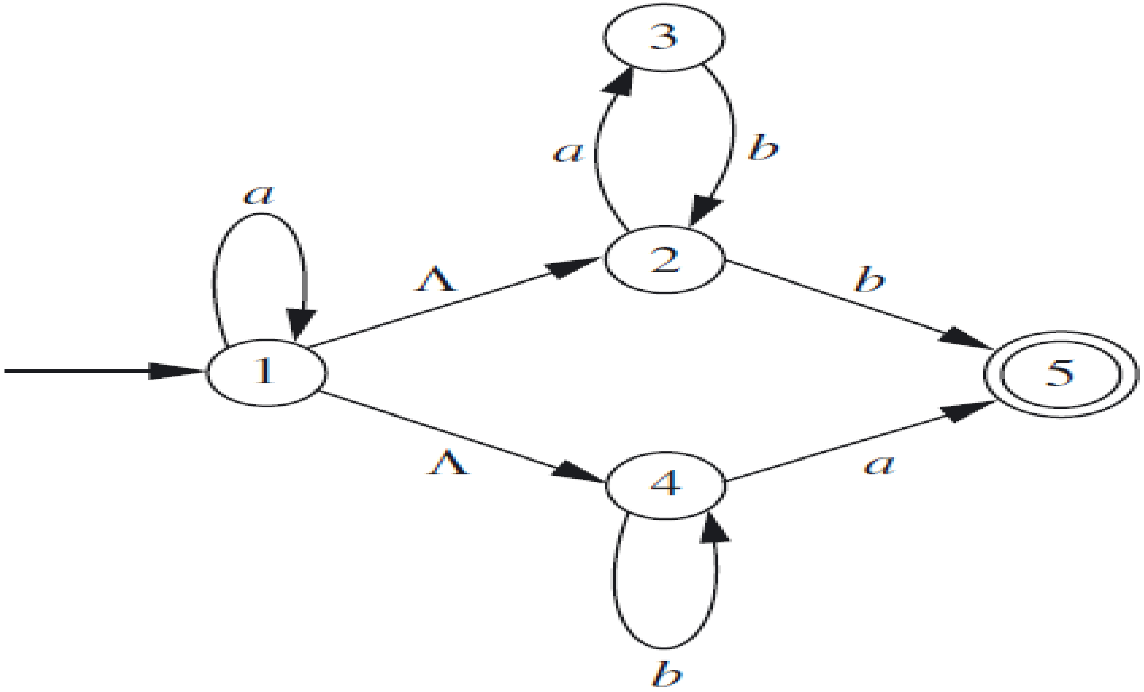
		a			b	
	λ		λ	λ		λ
1	{1,2}	{2,3}	{2,3}	{1,2}	Φ	Φ
2	{2}	{2,3}	{2,3}	{2}	Φ	Φ
3	{3}	Φ	Φ	{3}	{4}	{4,1,2}
4	{4,1,2}	{2,3}	{2,3}	{1,2,4}	{5}	{5}
5	{5}	{4}	{1,2,4}	{5}	Φ	Φ



NFA-λ

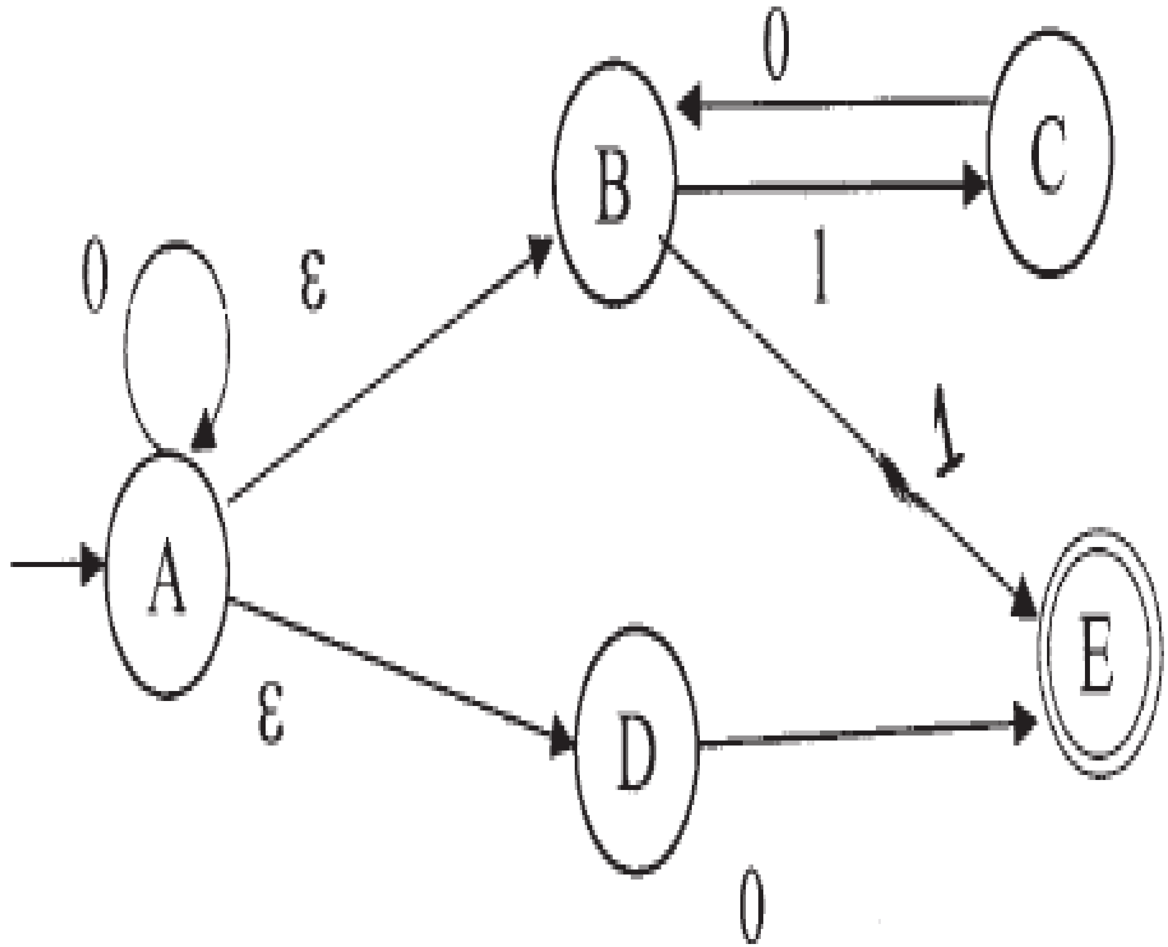
NFA





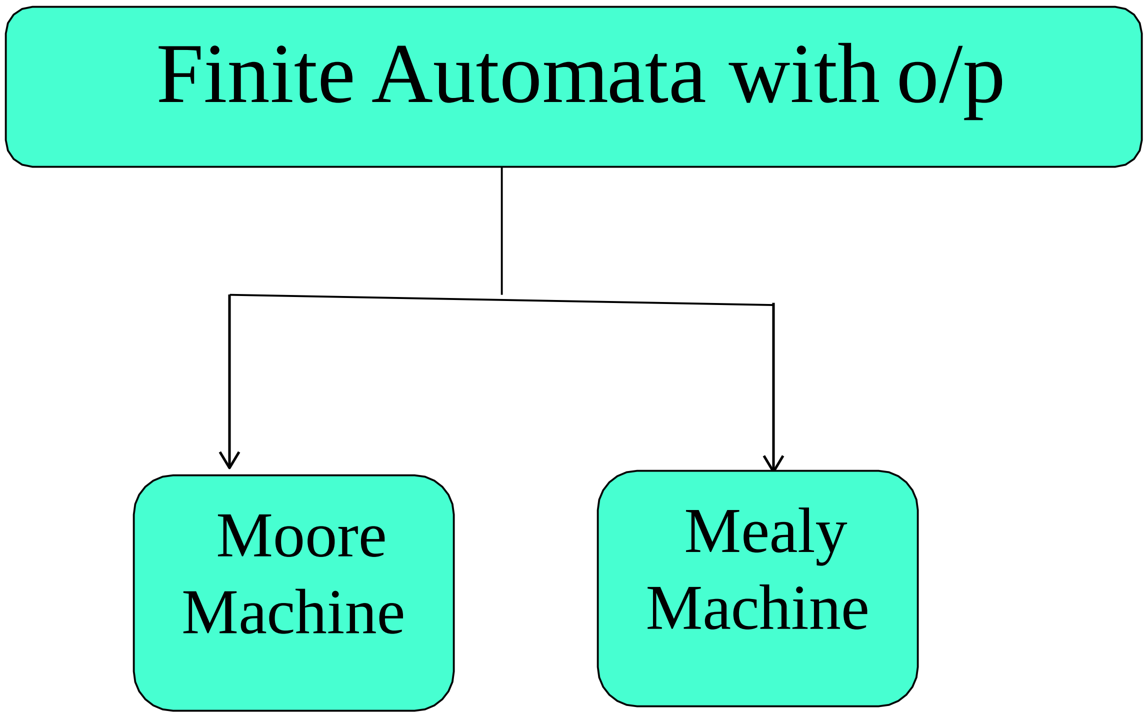
		a			b	
	λ		λ	λ		λ
1	{1,2,4},	{1,3,5}	{1,2,3,4,5}			
2						
3						
4						
5						

Convert the given NFA–ε to an NFA.



Review

- Minimization of DFA
- DFA, NFA and NFA- λ Equivalency


Finite Automata with o/p
Moore Machine
Mealy Machine

•Moore Machine and Mealy machine

Definition:-

$$(Q, \Sigma, q_0, \delta, \Delta, \gamma)$$

Where,

Q=set of states, Σ=i/p alphabet, q0= start/initial state,

$\delta = Q \times \Sigma \rightarrow Q$ (transition function),

Δ=o/p alphabet, γ =o/p function

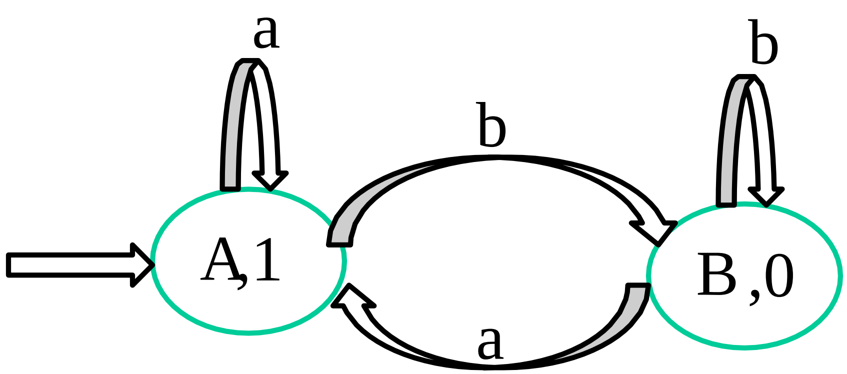
•Difference between Mealy and Moore machine is γ (o/p function)

γ (o/p function) for Moore machine is $\gamma : Q \rightarrow \Delta$
(outputs depend on only the present state)

γ(o/p function) for Mealy machine is $\gamma : Q \times \Sigma \rightarrow \Delta$
(Output depends on the present state as well as the present input)

- Moore and Mealy Machines are equally powerful

• Moore Machine

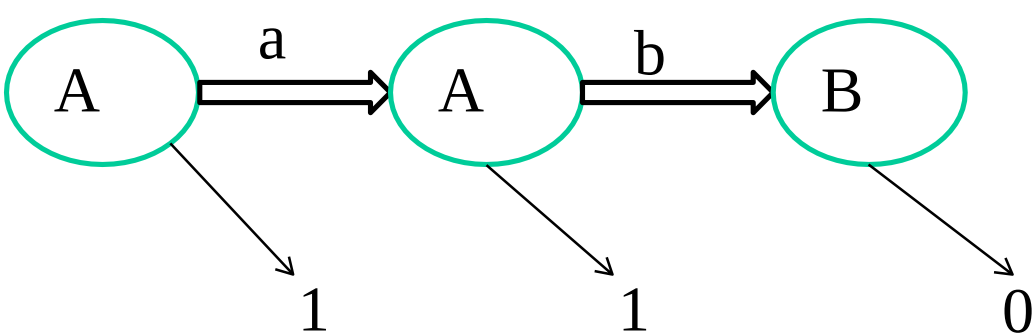


	Input		Output
	a	b	
A	A	B	1
B	A	B	0

$$\lambda: Q \rightarrow \Delta$$

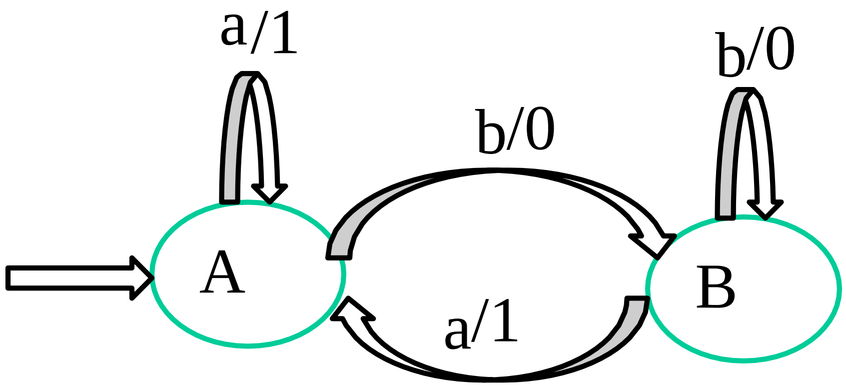
A→1 B→0

• string 'ab' on machine



- Applying string 'ab' on machine (as input) , got the output as 110.
- If number of input symbol is N then Number of output symbol is N+1.

• Mealy Machine



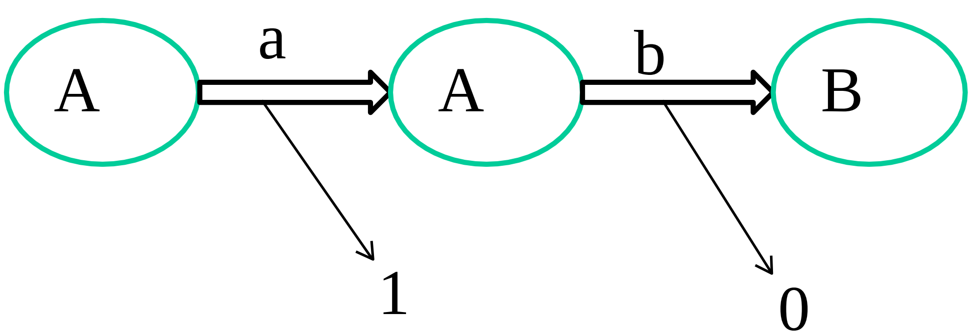
	Input	Output	Input	Output
	a		b	
A	A	1	B	0
B	A	1	B	0

$$\lambda: Q \times \Sigma \rightarrow \Delta$$

A,a → 1   A,b → 0

B,a → 1   B,b → 0

• string 'ab' on machine

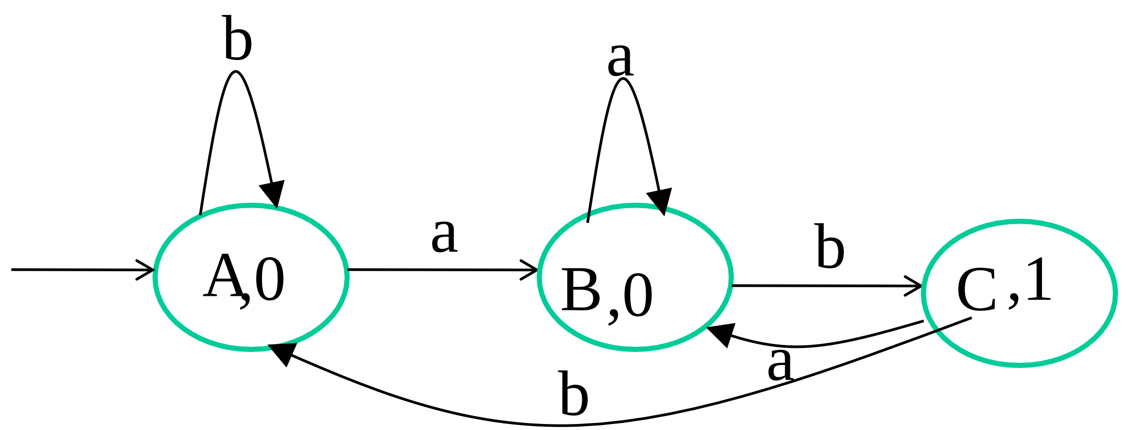


- Applying string 'ab' on machine (as input) , got the output as 10.
- If number of input symbol is N then Number of output symbol is N.

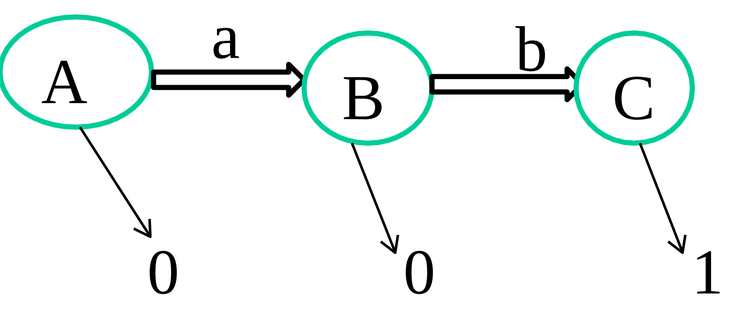
Example on Moore Machine

Construct a Moore Machine that take the set of all strings over {a, b} as i/p and prints '1' as o/p for every occurrence of ' ab' as a substring

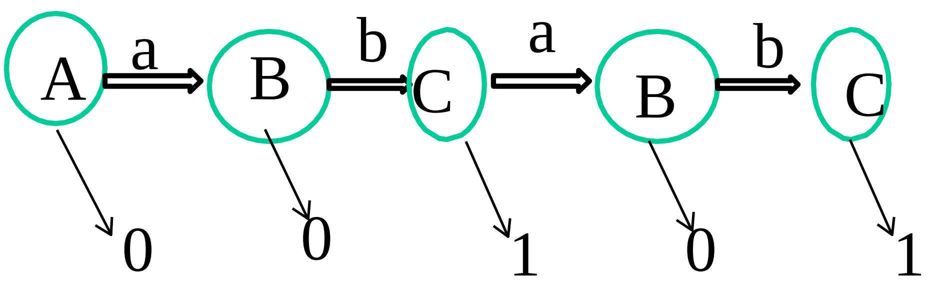
$Q=\{A, B, C\}$ $\Sigma=\{a, b\}$ $\Delta=\{0,1\}$



•string 'ab' on machine



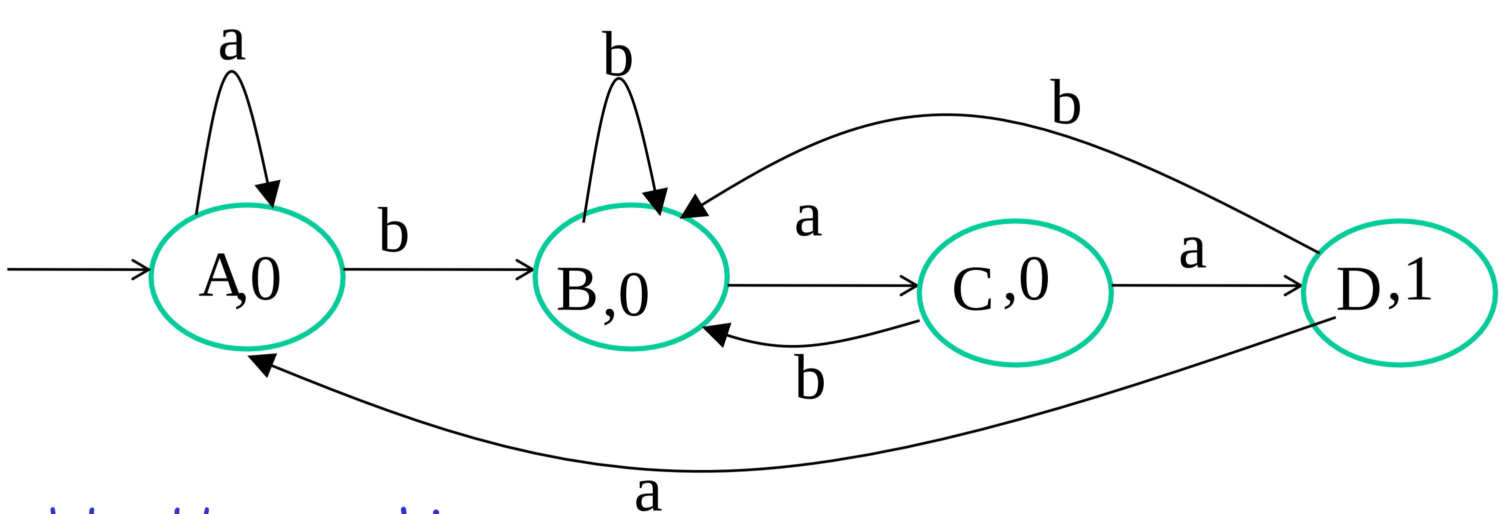
•string 'abab' on machine



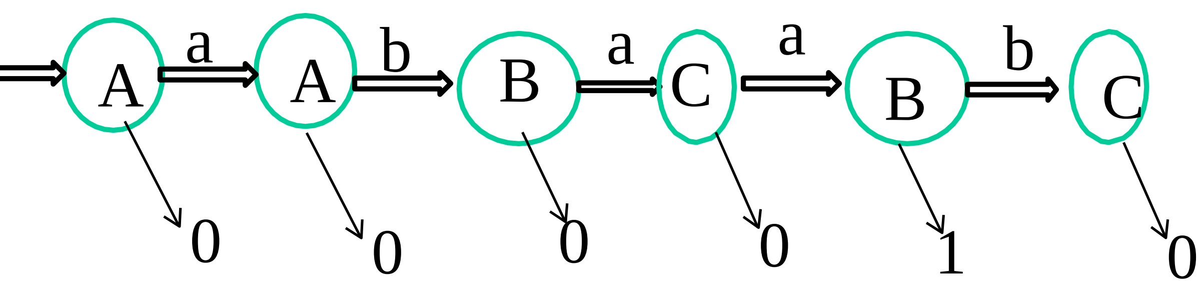
Example on Moore Machine

Construct a Moore Machine that take the set of all strings over {a, b} as i/p and counts no of occurrences of substring 'baa'

$Q=\{A, B, C\}$ $\Sigma=\{a, b\}$ $\Delta=\{0,1\}$



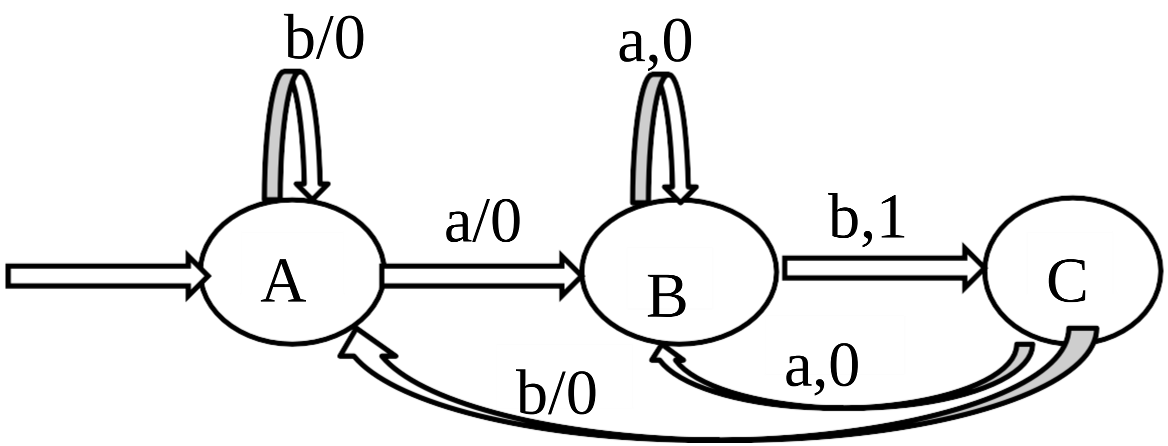
• string 'abaab' on machine



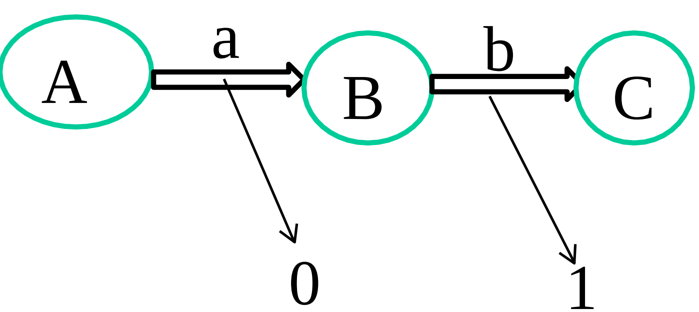
Example on Mealy Machine

Construct a Mealy Machine that take the set of all strings over {a, b} as i/p and prints '1' as o/p for every occurrence of ' ab' as a substring

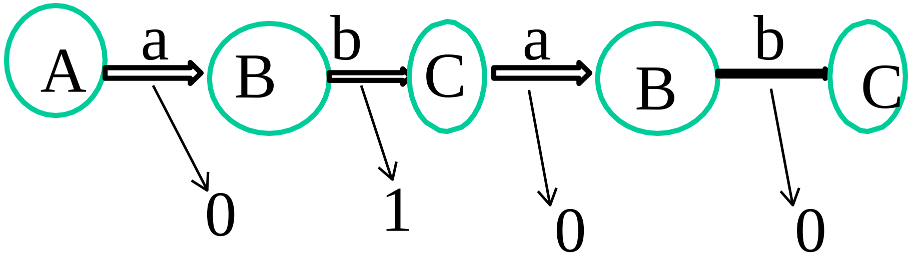
$$Q=\{A, B, C\}, \Sigma=\{a, b\}, \Delta=\{0,1\}$$



• string 'ab' on machine



• string 'abab' on machine



Conversion of Moore and Mealy Machine

Moore Machine ⟷ Mealy Machine

Moore Machine ⟶ Mealy Machine

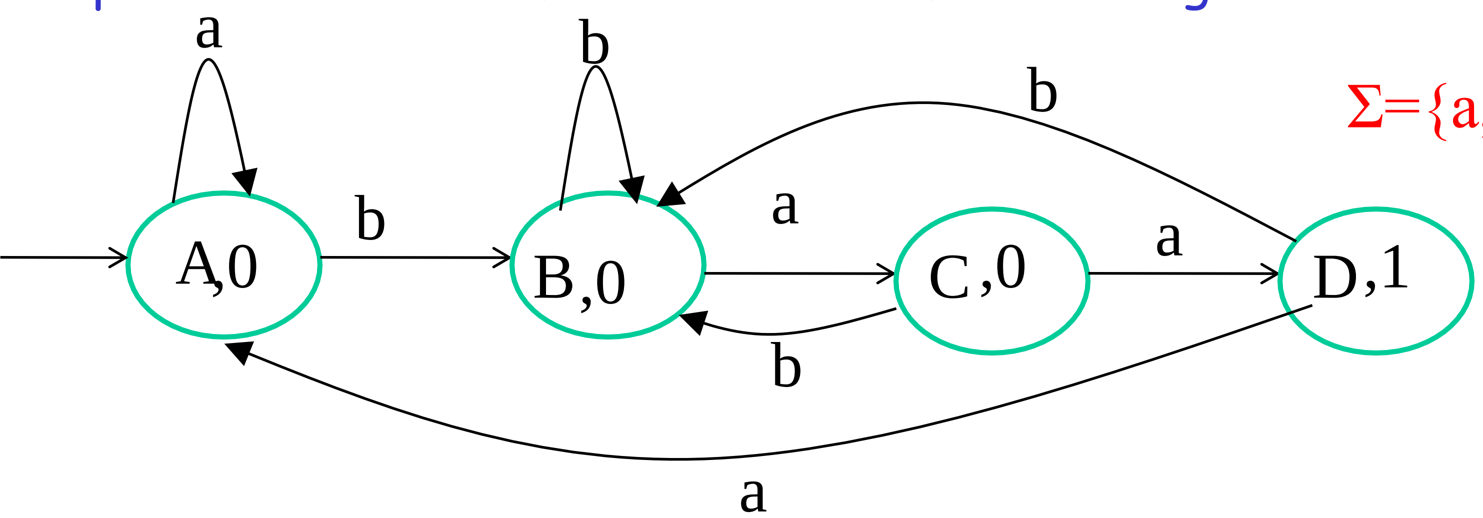
Mealy Machine ⟶ Moore Machine

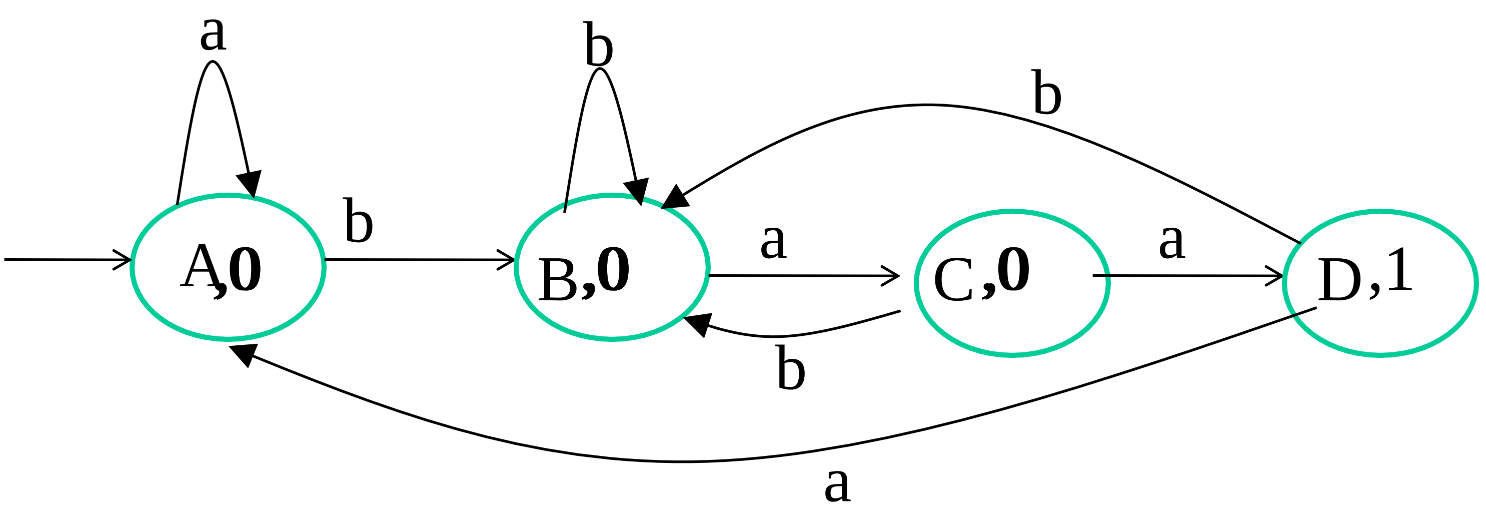
Moore Machine to Mealy Machine

Construct a Moore Machine that take the set of all strings over {a, b} as i/p and counts no of occurrences of substring 'baa'

$Q=\{A, B,C\}$

$\Sigma=\{a, b\}$ $\Delta=\{0,1\}$



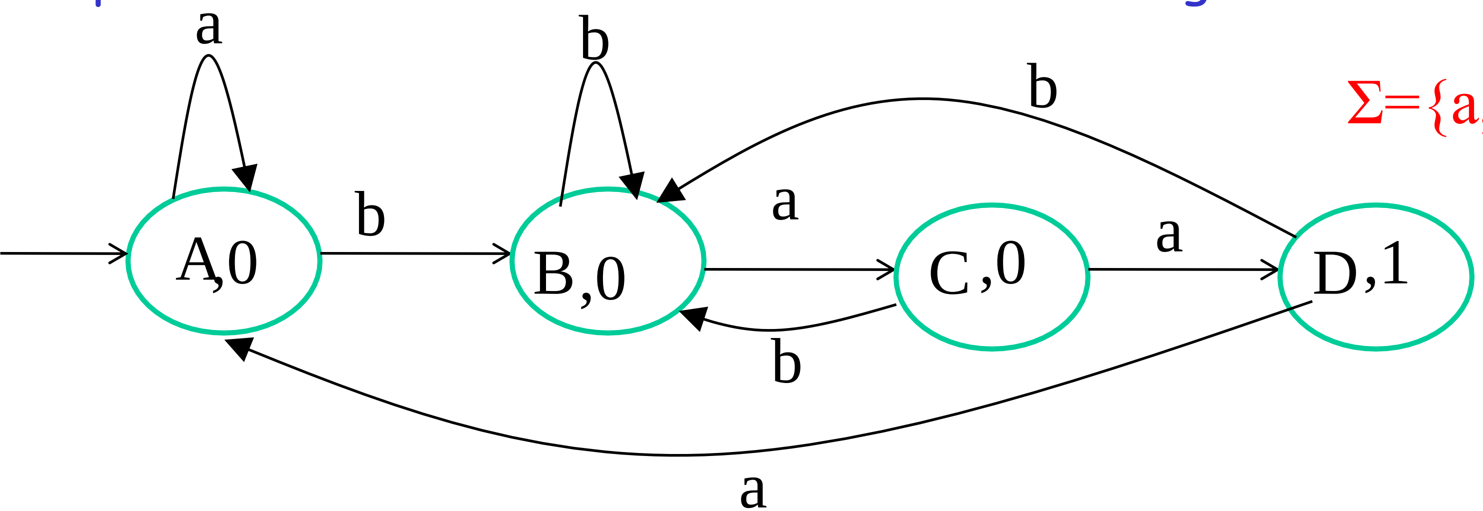


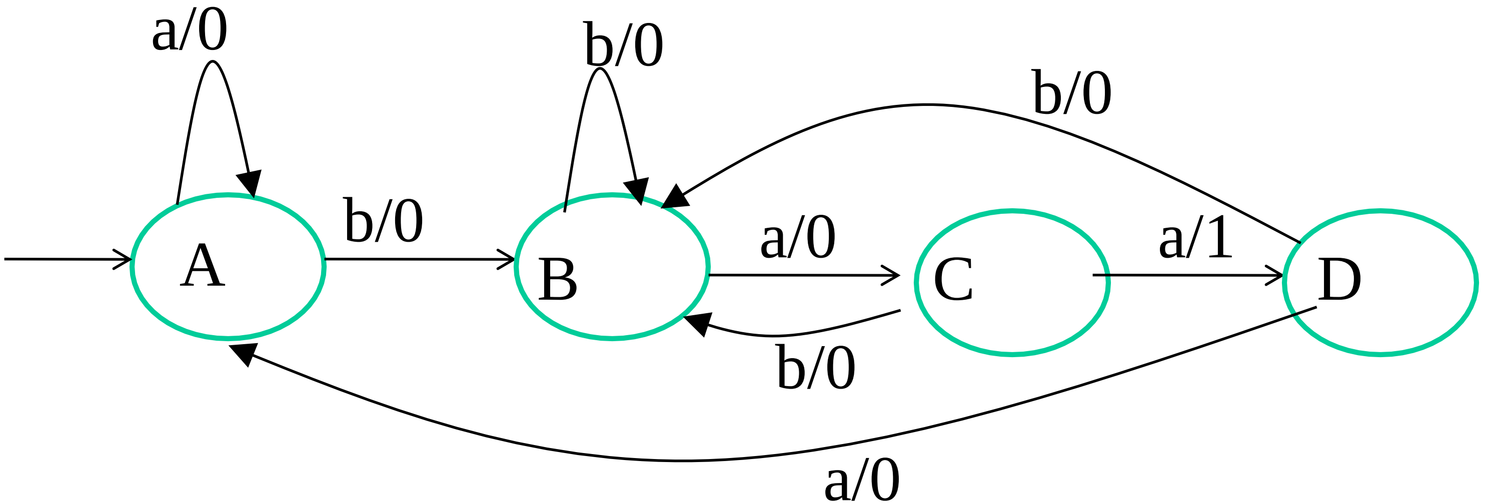
Moore Machine to Mealy Machine

Construct a Moore Machine that take the set of all strings over {a, b} as i/p and counts no of occurrences of substring 'baa'

Q={A, B,C}

Σ={a, b}　　Δ={0,1}

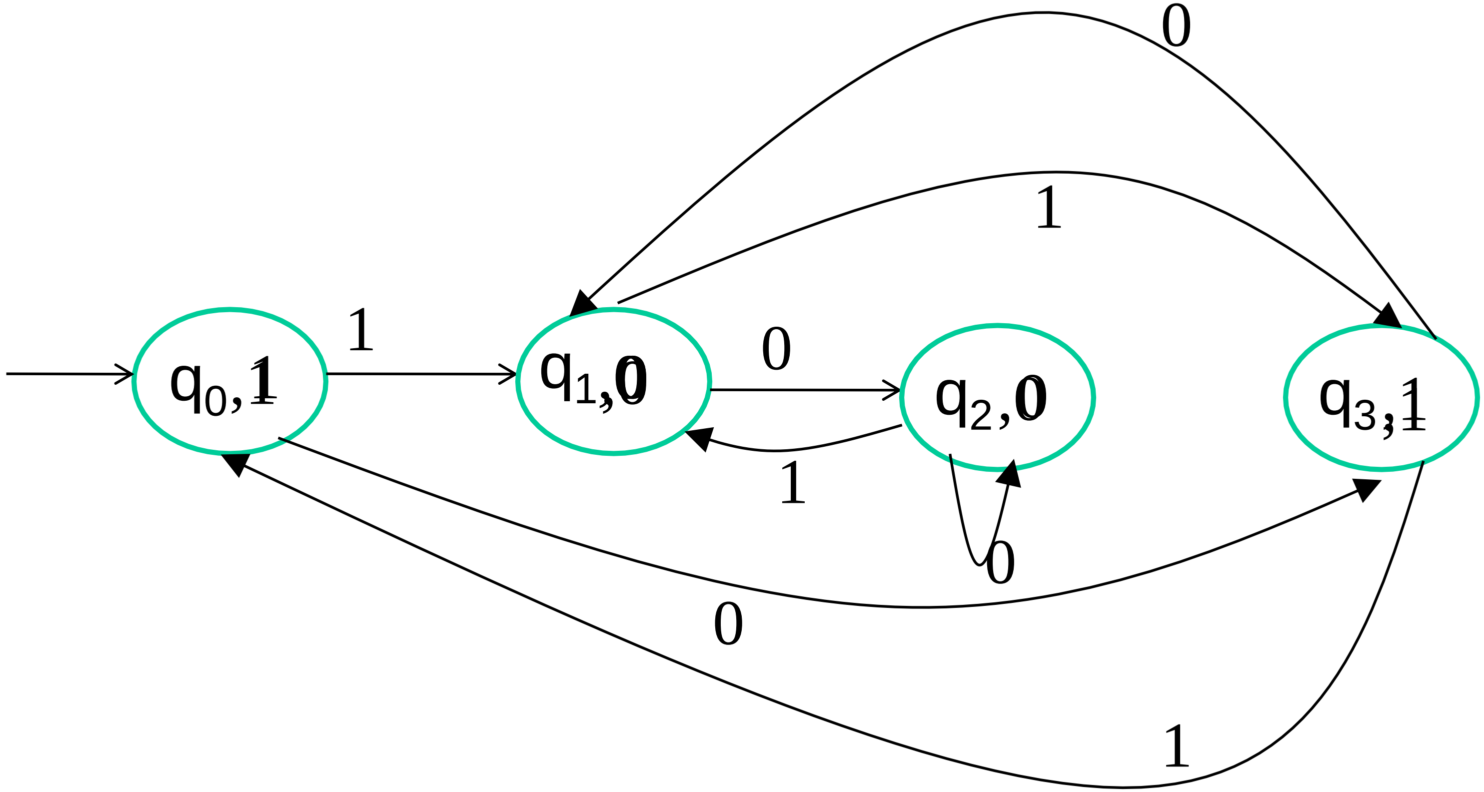




Convert Following Moore Machine into Mealy Machine

Aug-2015
INSEM

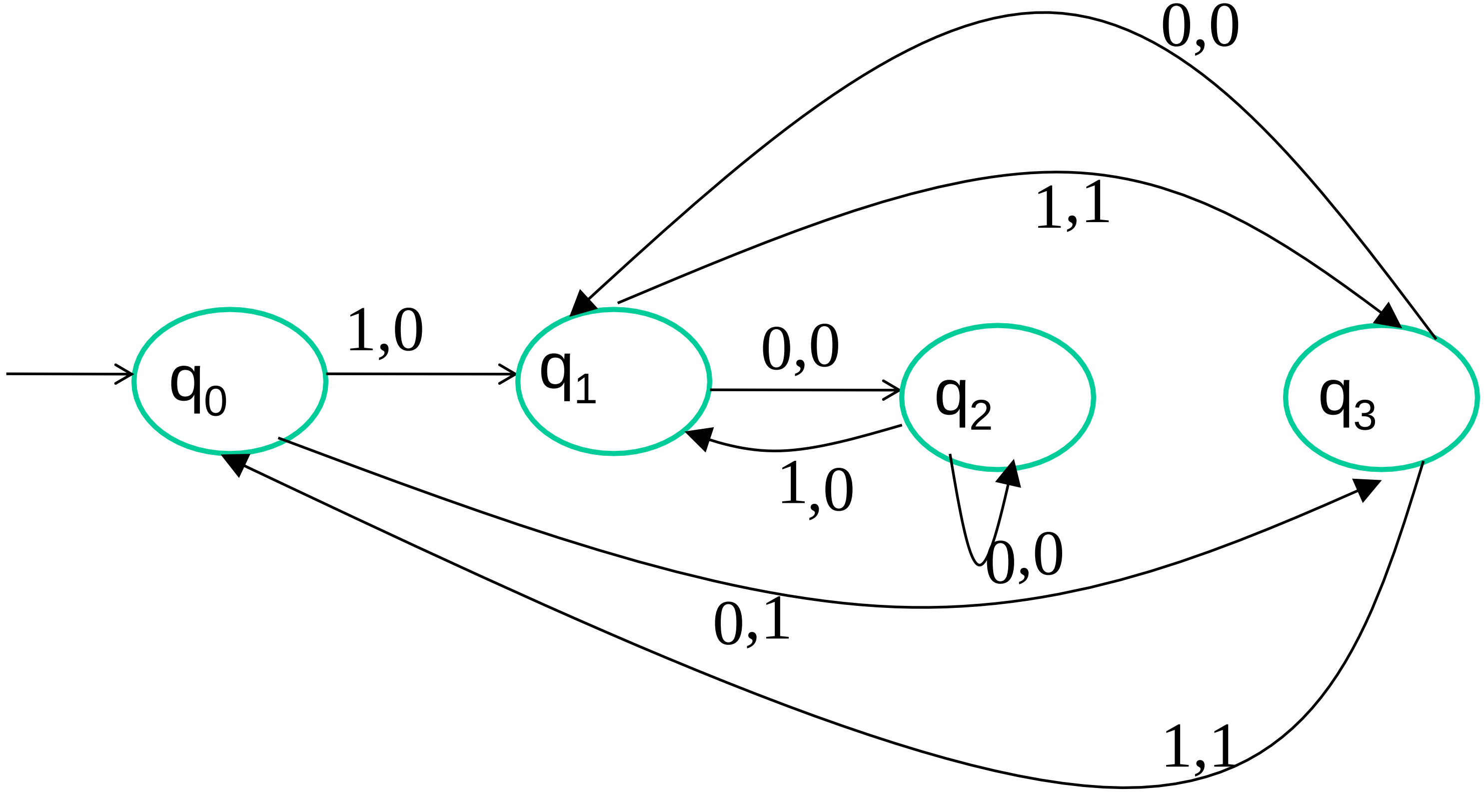
Present State	Next State		Output
	a = 0	a = 1	
-> q0	q3	q1	1
q1	q2	q3	0
q2	q2	q1	0
q3	q1	q0	1



Convert Following Moore Machine into Mealy Machine

Present State	Next State		Output
	a = 0	a = 1	
-> q0	q3	q1	1
q1	q2	q3	0
q2	q2	q1	0
q3	q1	q0	1

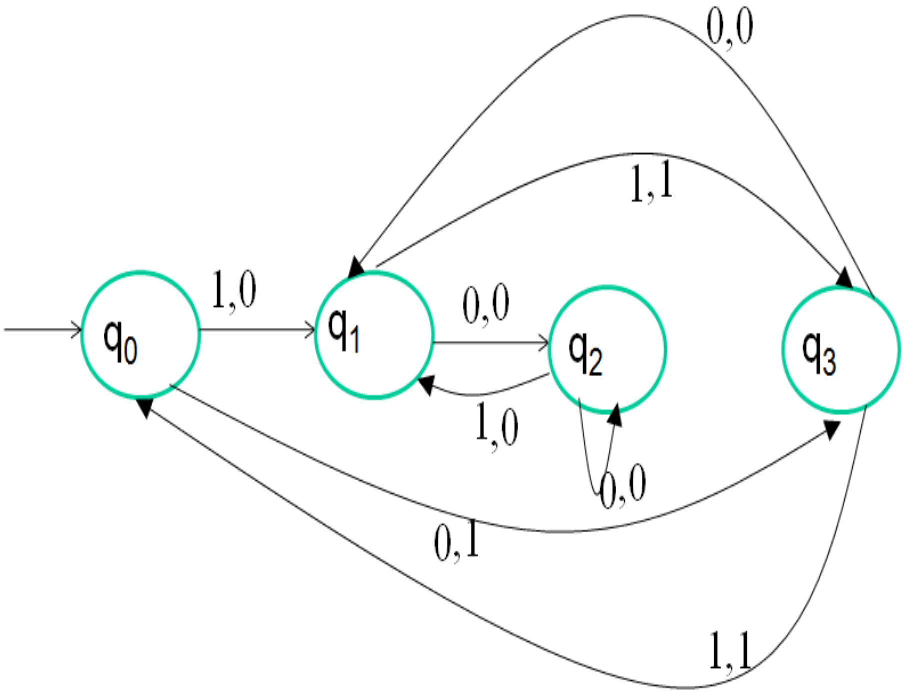
Aug-2015
INSEM



Convert Following Moore Machine into Mealy Machine

Aug-2015 INSEM

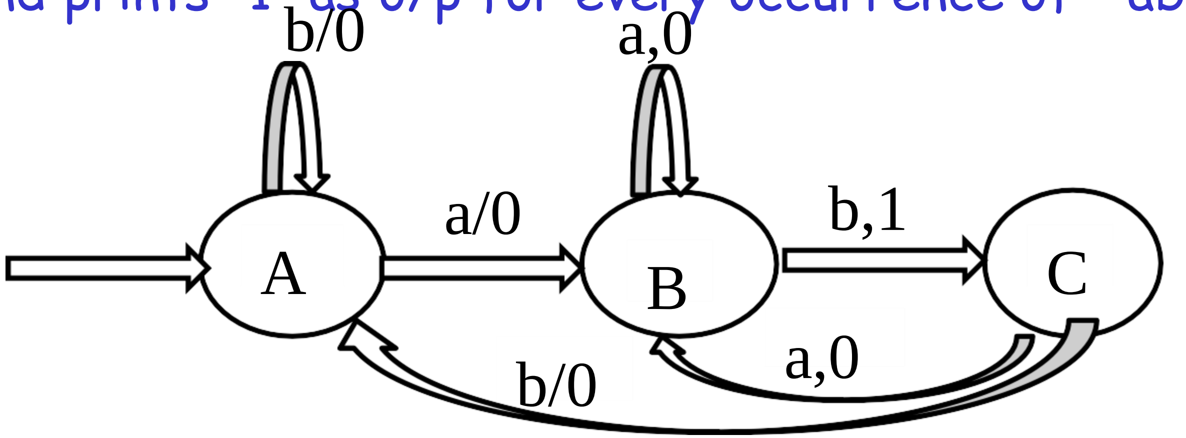
Present State	Next State		Output
	a = 0	a = 1	
-> q0	q3	q1	1
q1	q2	q3	0
q2	q2	q1	0
q3	q1	q0	1

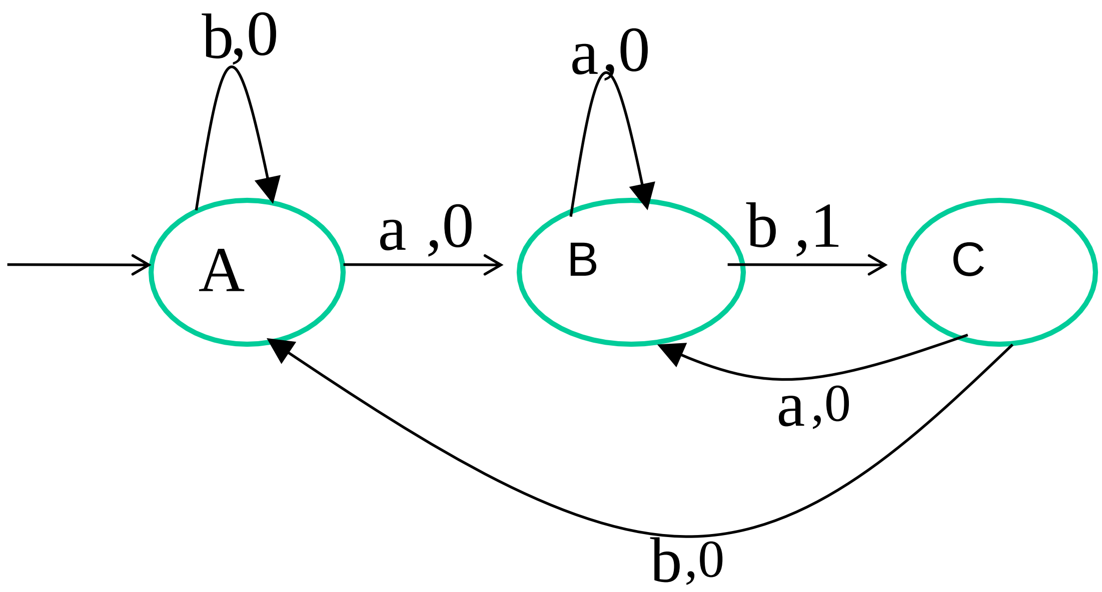


	Input	Output	Input	Output
	0		1	
q_0	q_3	1	q_1	0
q_1	q_2	0	q_3	1
q_2	q_2	0	q_1	0
q_3	q_1	0	q_0	1

Mealy Machine to Moory Machine

Construct a Mealy Machine that take the set of all strings over {a, b} as i/p and prints '1' as o/p for every occurrence of ' ab' as a substring

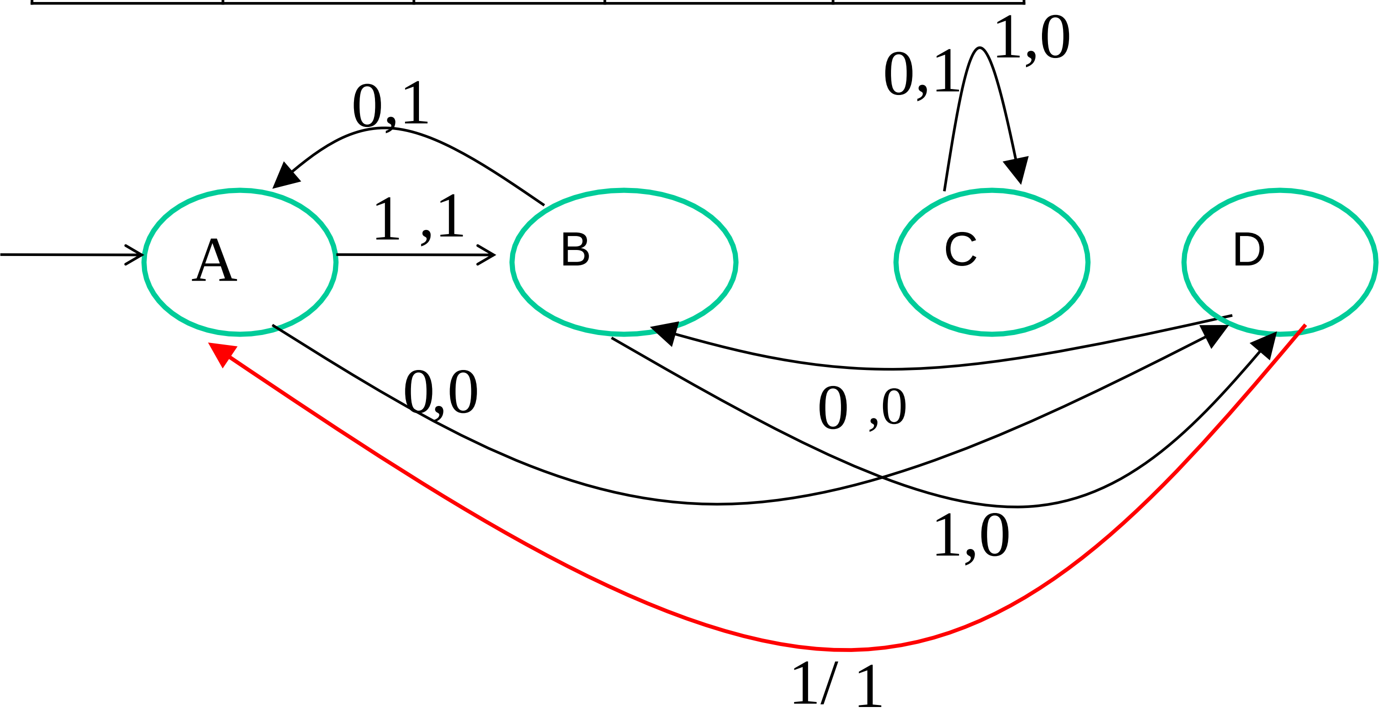


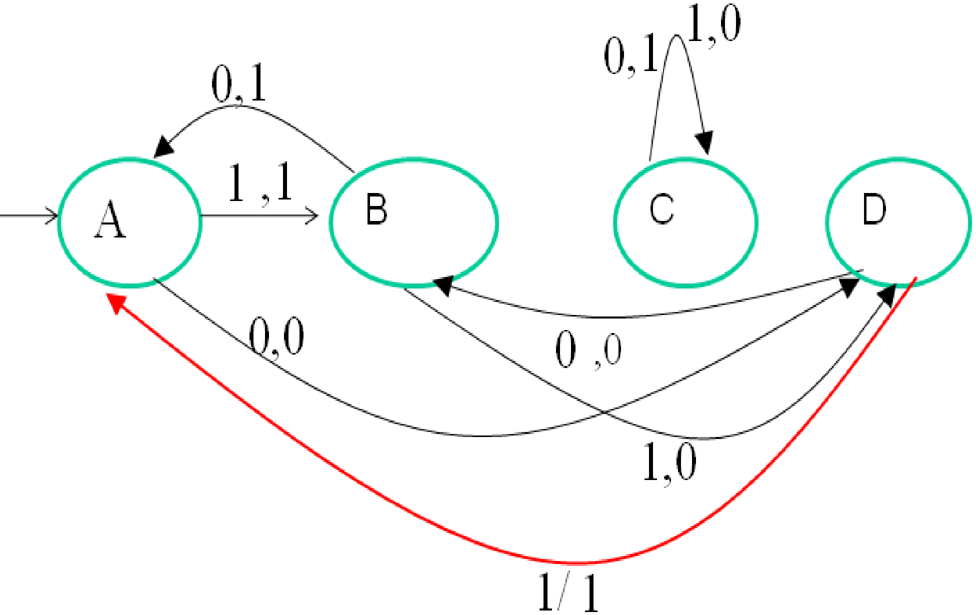


Convert Following Mealy Machine into Moore Machine

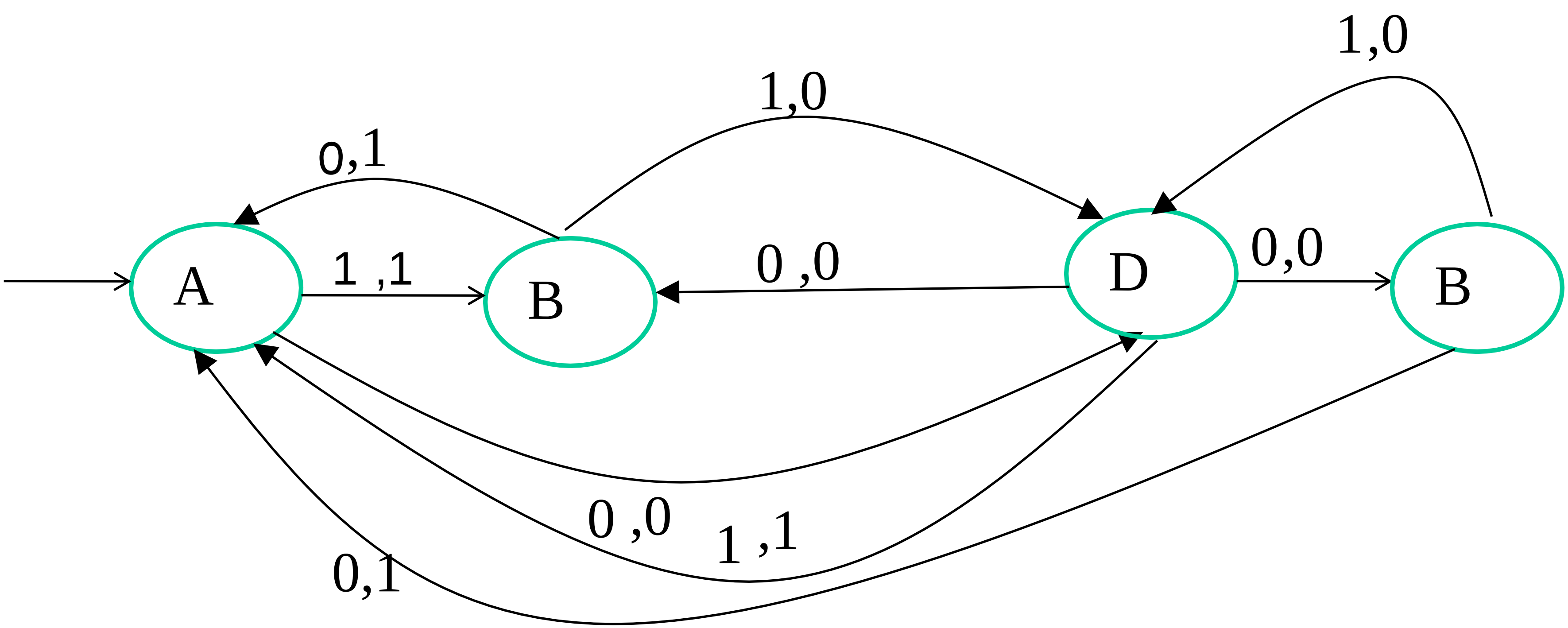
Aug-2015
INSEM

Present State	0		1	
	Next State	Output	Next State	Output
→ A	D	0	B	1
B	A	1	D	0
C	C	1	C	0
D	B	0	A	1





Present State	0		1	
	Next State	Output	Next State	Output
→ A	D	0	B	1
B	A	1	D	0
C	C	1	C	0
D	B	0	A	1



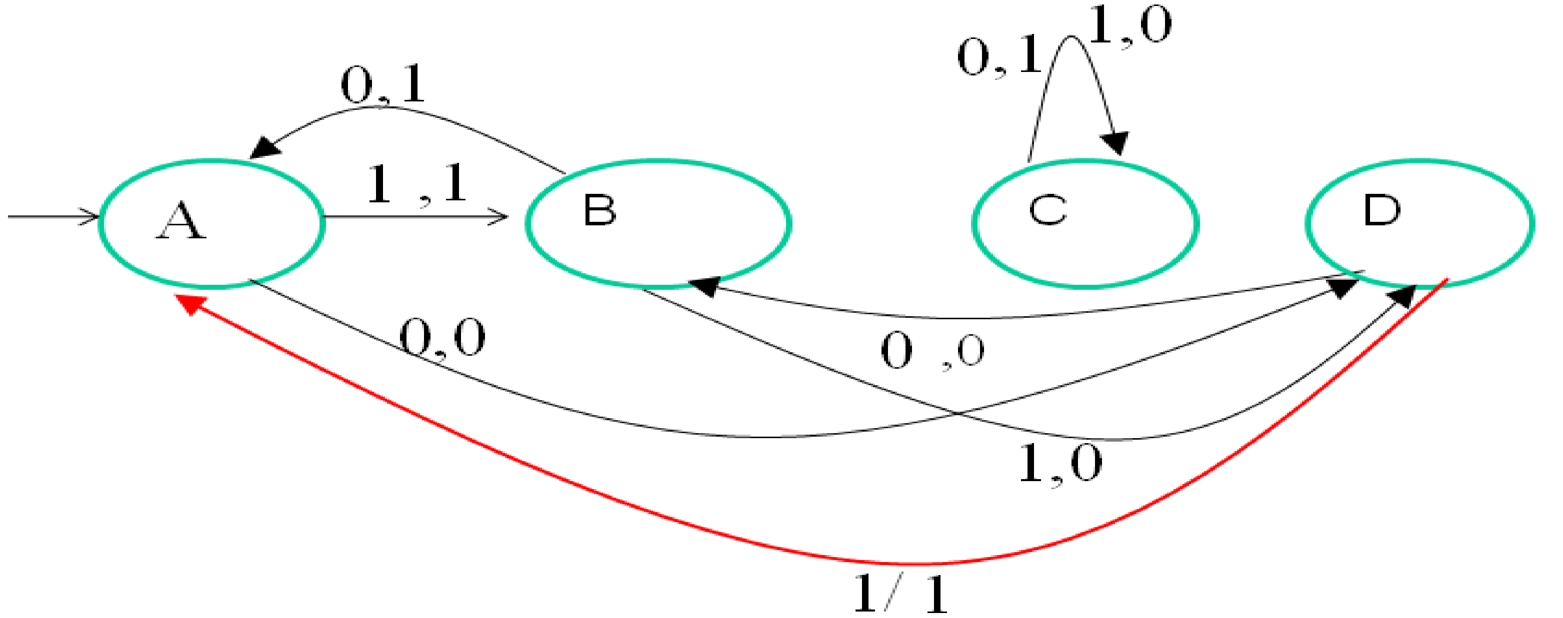
Review

- FA With Output
- Moore Machine and Mealy Machine
- Equivalency of Moore and Mealy Machine
- Conversion of Moore and Mealy Machine

Convert Following Mealy Machine into Moore Machine

Aug-2015
INSEM

Present State	0		1	
	Next State	Output	Next State	Output
→ A	D	0	B	1
B	A	1	D	0
C	C	1	C	0
D	B	0	A	1



Present State	0		1	
	Next State	Output	Next State	Output
→ A	D	0	B	1
B	A	1	D	0
C	C	1	C	0
D	B	0	A	1

Present State	Input		
	0	1	Output
A	D	B1	1
D	B0	A	0
B0	A	D	0
B1	A	D	1
C0	C1	C0	0
C1	C1	C0	1

b) Construct Mealy machine equivalent to the given Moore machine

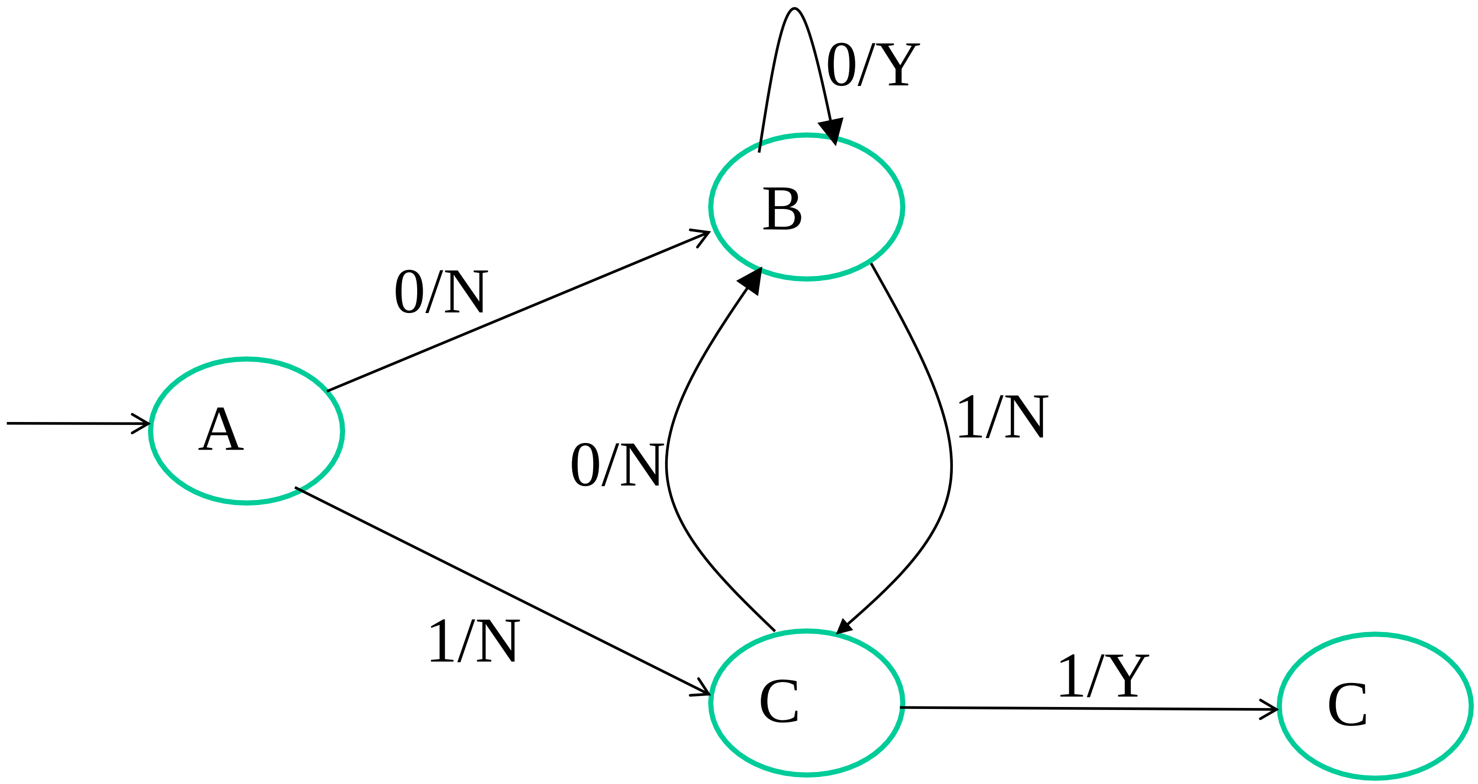
Aug-2017
INSEM

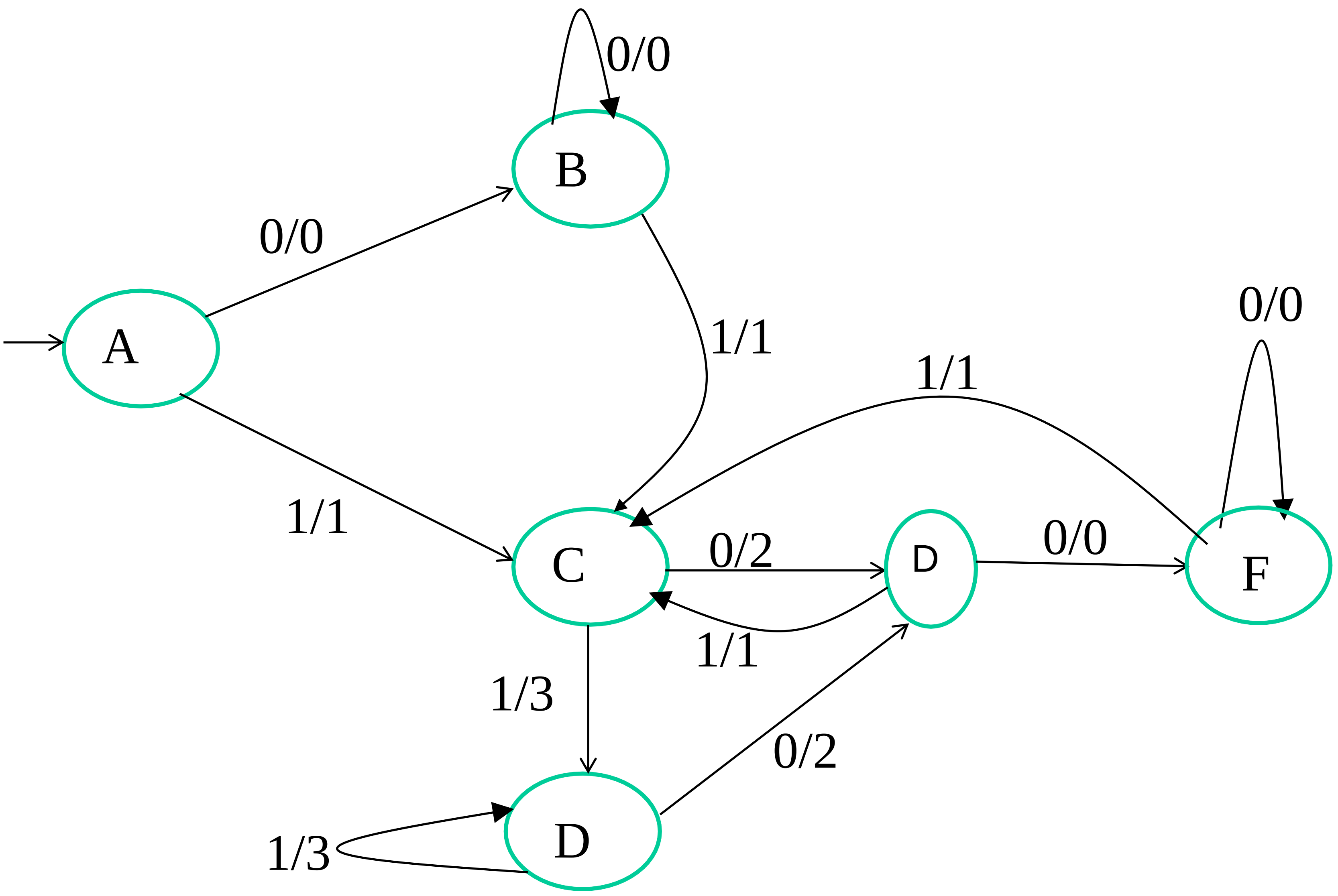
	0	1	O/P
q0	q0	q1	N
q1	q0	q2	N
q2	q0	q3	N
q3	q0	q3	Y

Start state : q0 ; Final state : q3

Design Moore Machine for divisibility by 3 tester for binary number. (Nov-2017 6 Marks)

Convert following Mealy Machine to Moore Machine (Nov-2017 6 Marks)





0/0
B
0/0
A
1/1
1/1
0/0
1/1
C
0/2
D
0/0
F
1/1
1/3
0/2
1/3
D

Thank you!!!!!!!