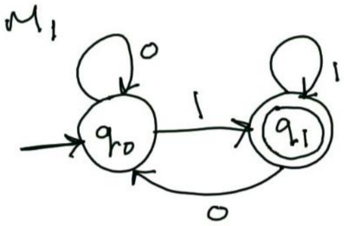
Non-deterministic Finite Automaton (Lecture 6)

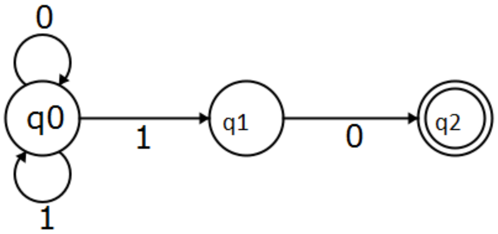
Why NFA is called Non-deterministic?

- Deterministic machine
  - When the machine is at a given state, for a specific input, its gives one output at each time.



- Non-Deterministic machine
  - When the machine is at a given state, for a specific input, it may give more than one output at each time.
  - Every DFA consider NFA

- "Deterministic" means "if you put the system in the same situation twice, it is guaranteed to make the same choice both times".
- "Non-deterministic" means "not deterministic", or in other words, "if you put the system in the same situation twice, it might or might not make the same choice both times".
- A non-deterministic finite automaton (NFA) can have multiple transitions out of a state. This means there are multiple options for what it could do in that situation. It is not forced to always choose the same one; on one input, it might choose the first transition, and on another input it might choose the same transition.
- Take this automaton for instance, it's an NFA and it accepts the string 0110. it accepts strings that end in 10.

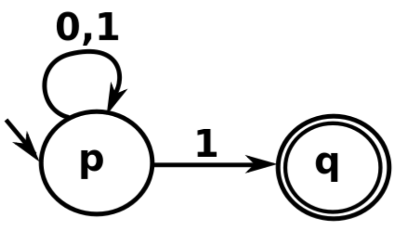


To see that we just need to check whether it reaches an accept state.

$q0 \rightarrow 1$
$q0 \rightarrow 0$
$q1 \rightarrow 1$
$q2 \rightarrow 0$

Now in the red line there was another possibility, that is when reading the second ***1***. We could stay in ***q*0** and then stay in ***q0*** when reading the last ***0***. Automata have no memory, so there's no way to 'save' a state and check later if my string ends with ***10***, it's like this NFA it's making a guess whether the string ends with ***10*** before branching to an acceptable state. The nondeterminism here is making many choices and always making the right ones.

Example



On the image above, when we are dealing with string "**00111**", notice that when encountering the first "**1**", there are two possible ways to follow. One can stay at "**p**" or go to "**q**". If the automata was to move to the "**q**", it wouldn't accept the string (since there are no edges coming out of the "**q**"). But the string can be accepted by this automata by going to the "**q**" with only the last **1**, while staying at "**p**" for everything else. RE= (0+1)*1

- In a **NFA**, for each state there can be zero, one, two, or more transitions corresponding to a particular symbol.
- **Nondeterministic** means it can transition to, and be in, multiple states at once (i.e. for some given input). **Deterministic** means that it can only be in, and transition to, one state at a time (i.e. for some given input).
- If **NFA** gets to state with more than one possible transition corresponding to the input symbol, we say it branches.
- If **NFA** gets to a state where there is no valid transition, then that branch dies.

► **NFA** has finite number of states; the machine is called **Non-deterministic Finite Machine** or **Nondeterministic Finite Automaton**.

Formal Definition of an NFA

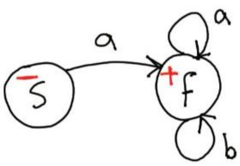
The formal definition of an NFA consists of a 5-tuple, in which order matters.

Similar to a DFA, the formal definition of NFA is: (Q, **Σ**, δ, q0, F), where

1. **Q** is a finite set of all states
2. **Σ** is a finite set of all symbols of the alphabet
3. **δ: Q x Σ** → **Q** is the transition function from state to state
4. **q0 ∈ Q** is the start state, in which the start state must be in the set Q
5. **F ⊆ Q** is the set of accept states, in which the accept states must be in the set Q

The only difference between an NFA and a DFA for their formal definitions is that for an NFA, you must specify the empty string (ε) within your delta function, along with the other symbols.

► The **NFA** with **epsilon**-transition is a finite state machine in which the transition from one state to another state is allowed without any input symbol i.e. empty string ε.



For our NFA above, the formal definition would be:

- **Q** → {s, f}
- **Σ** → {a, b}
- **Start state** → s
- **F** → { f }
- **δ functions:**

  δ(s,a) = {f}
  δ(s,b) = { }
  δ(s,ε) = { }
  δ(f,a) = {f}
  δ(f,b) = {f}
  δ(f,ε) = { }

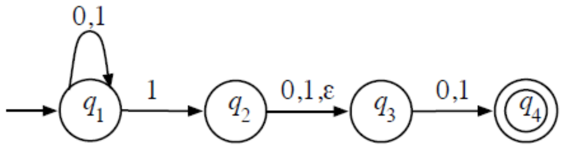
The nondeterministic finite automaton is a variant of finite automaton with two characteristics:

• ϵ-transition: state transition can be made without reading a symbol;

• Nondeterminism: zero or more than one possible value may exist for state transition.

An Example Nondeterministic Finite Automaton

An NFA that accepts all strings over {0, 1} that contain a 1 either at the third position from the end or at the second position from the end.



• There are two edges labeled 1 coming out of q_1.

• There are no edges coming out of q_4.

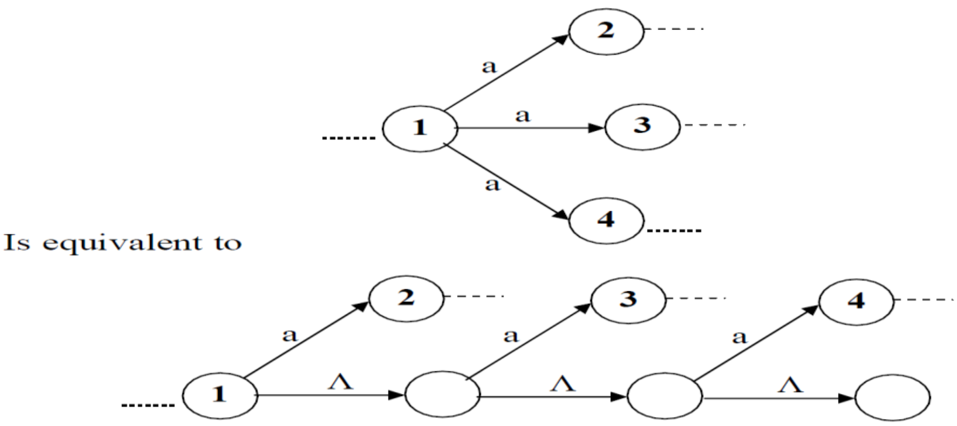
• The edge from q_2 is labeled with ε, in addition to 0 and 1.

Graphical Representation of an NDFA: (same as DFA)

An NDFA is represented by digraphs called state diagram.

- The vertices represent the states.
- The arcs labeled with an input alphabet show the transitions.
- The initial state is denoted by an empty single incoming arc or -.
- The final state is indicated by double circles or +.

We can convert any NFA into a TG with no repeated labels from any single state as in the following:



Example

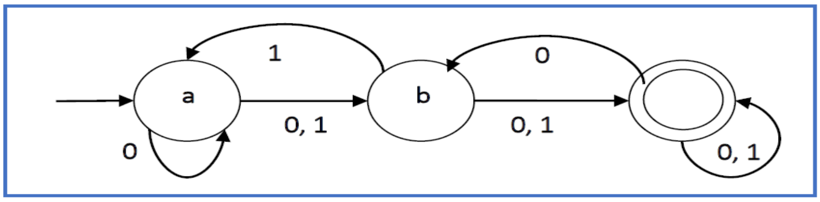
Let a non-deterministic finite automaton be:

- Q = {a, b, c}
- Σ = {0, 1}
- q0 = {a}
- F={c}

The transition function δ as shown below:

Present State	Next State for Input 0	Next State for Input 1
a	a, b	b
b	c	a, c
c	b, c	c

Its graphical representation would be as follows:



NDFA – Graphical Representation

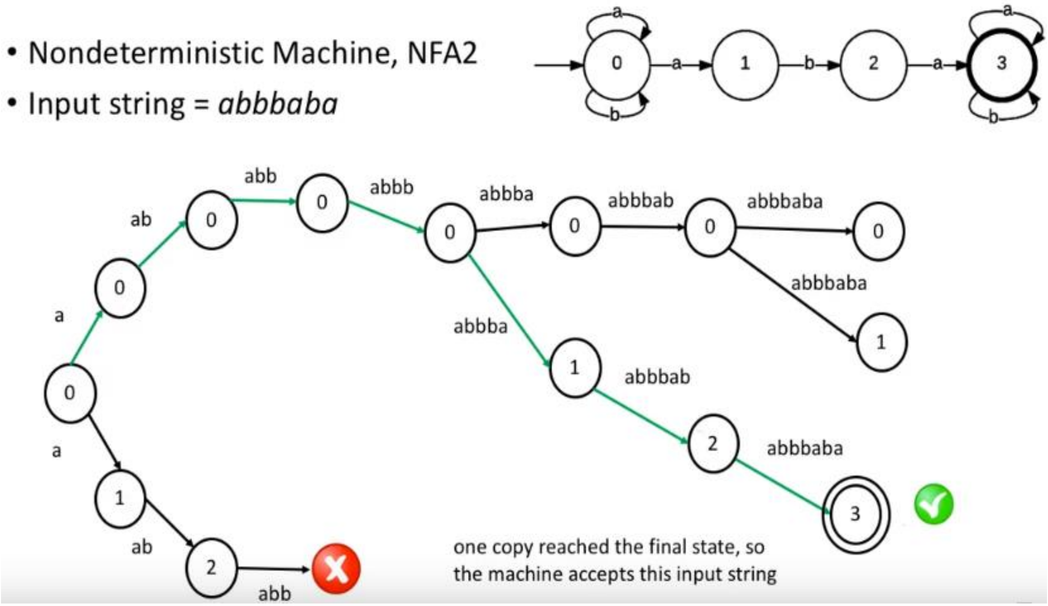
Any FA will satisfy the definition of an NFA. We have:

1. Every FA is an NFA.
2. Every NFA has an equivalent TG.
3. By Kleen's theorem, every TG has an equivalent FA.

Therefore:

Language of FA's ⊂ language of NFA's ⊂ language of TG's = language of FA's

Example



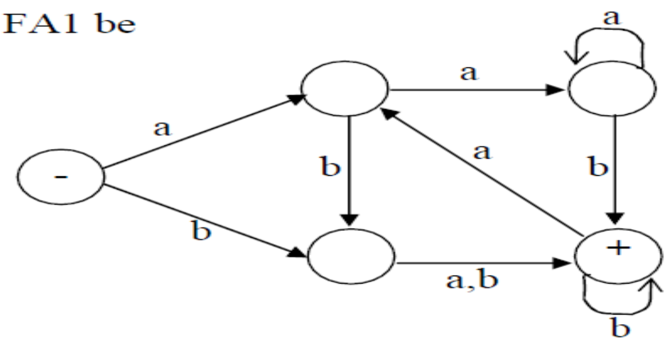
Theorem

FA = NFA

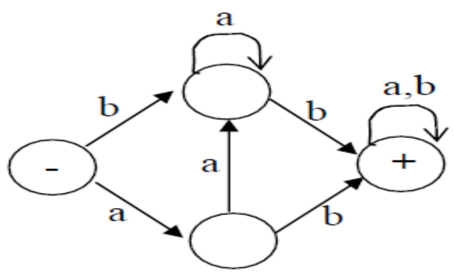
By which we mean that any language defined by a nondeterministic finite automaton is also definable by a deterministic (ordinary) finite automaton and vice versa.

<u>*Example*</u>

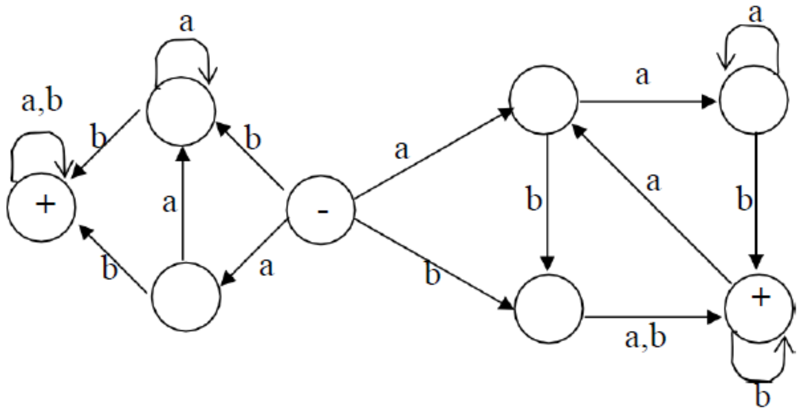
Let FA1 be



And let FA2 be



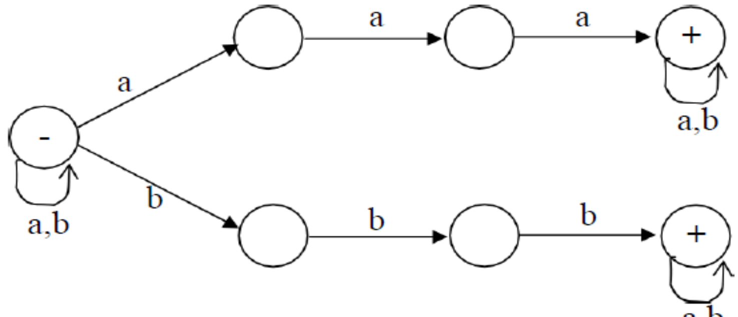
Then NFA3= FA1 + FA2 is



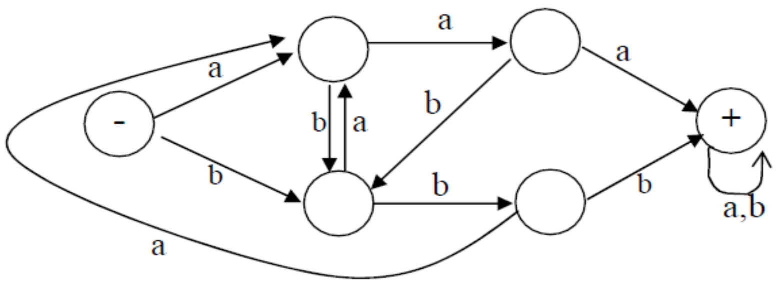
It is sometimes easier to understand what a language is from the picture of an NFA that accepts it than from the picture of an FA as in the following example.

Example

The NFA and FA below accepts the language of all words that contains either a triple a (the substring aaa) or a triple b (the substring bbb) or both.

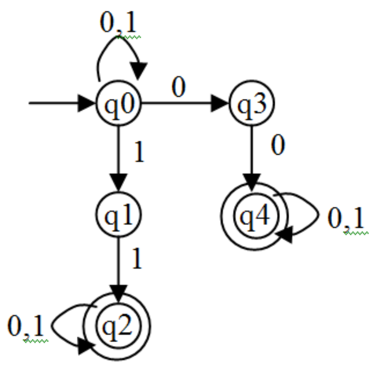


NFA

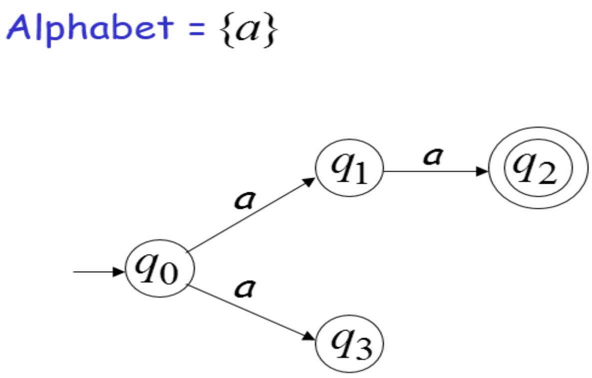


FA

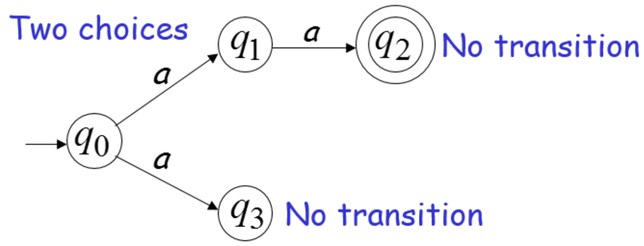
NDFSA Example



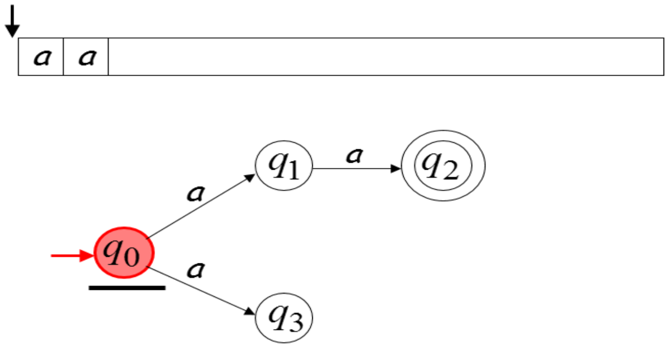
Example



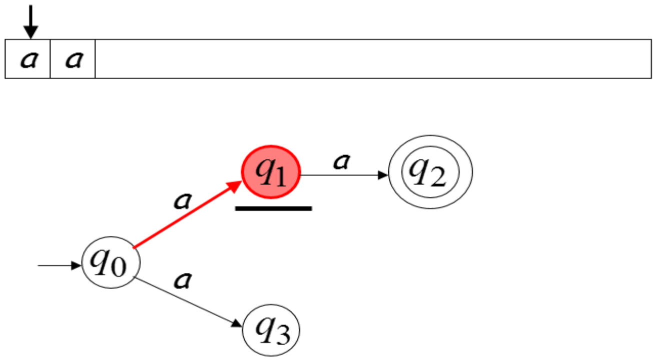
Alphabet = $\{a\}$



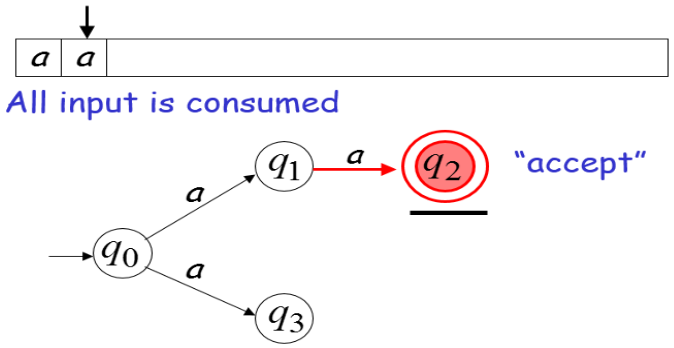
First Choice



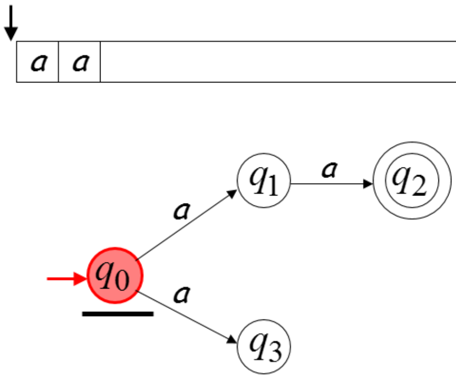
First Choice



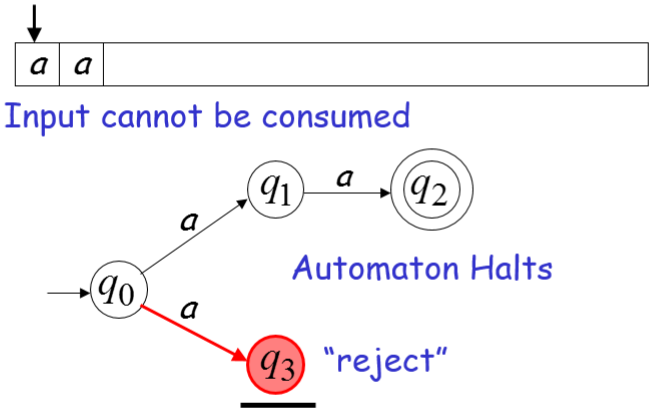
First Choice



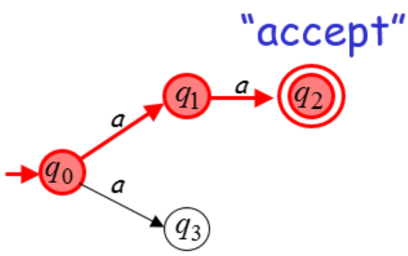
Second Choice



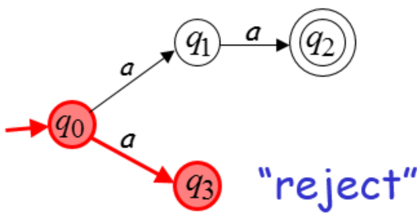
Second Choice



aa is accepted by the NFA:

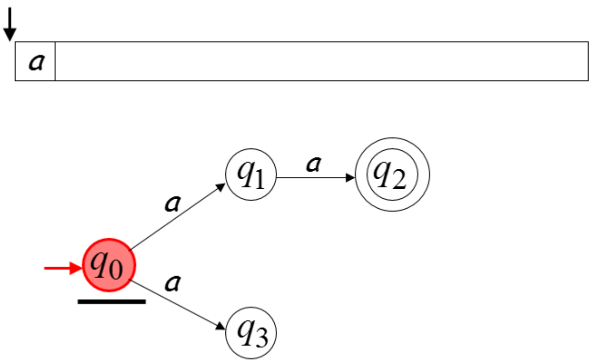


because this computation accepts aa

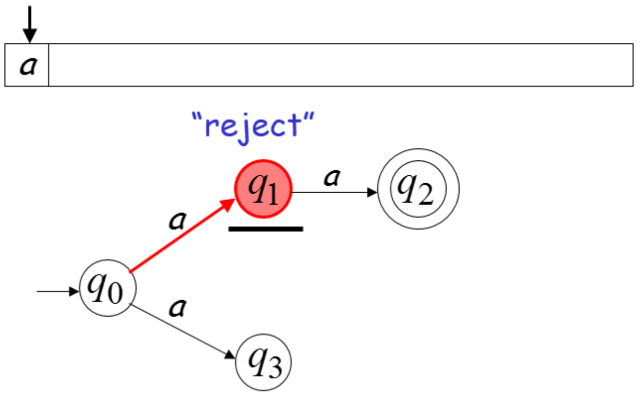


this computation is ignored

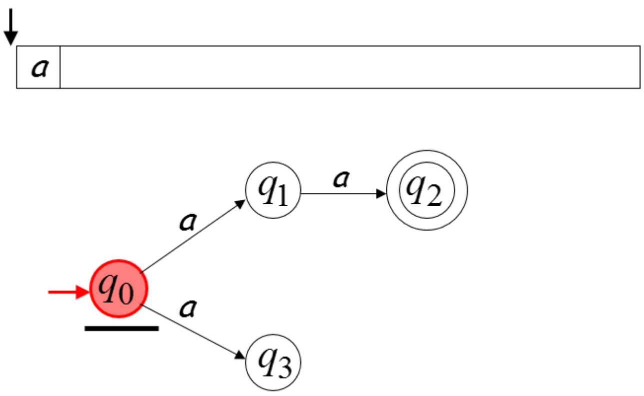
Rejection example



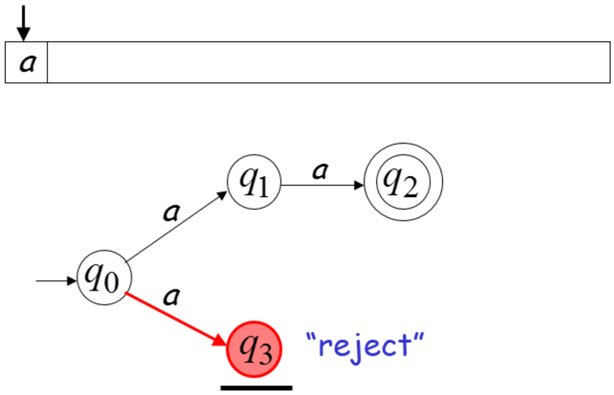
First Choice



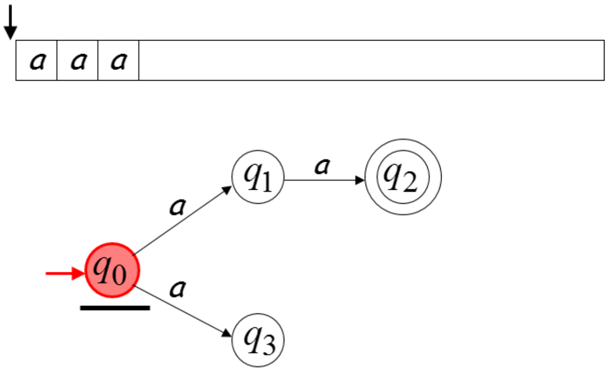
Second Choice



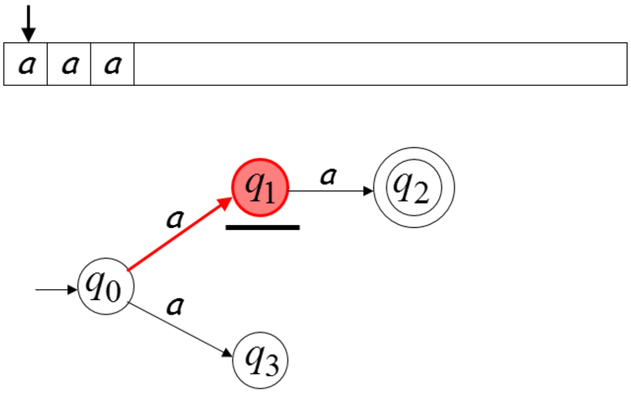
Second Choice



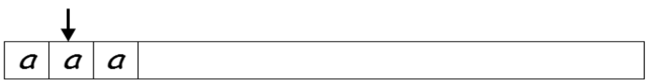
Another Rejection example



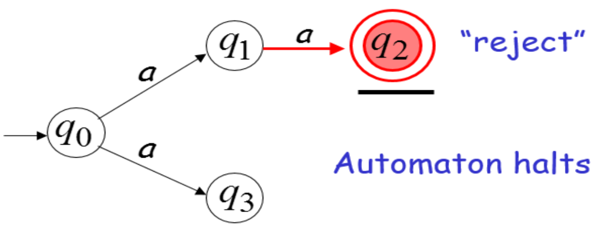
First Choice



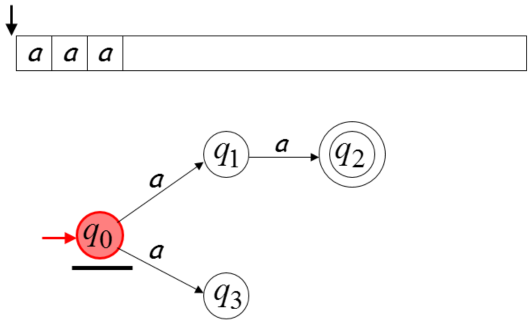
First Choice



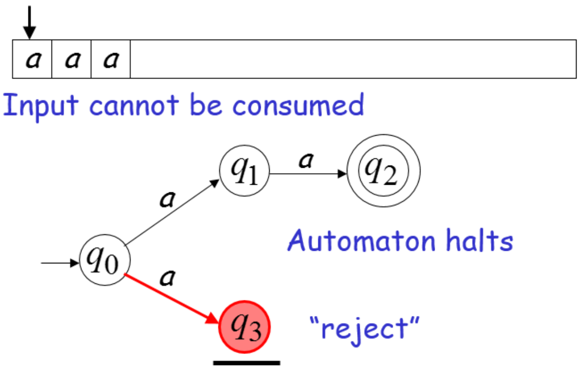
Input cannot be consumed



Second Choice



Second Choice



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

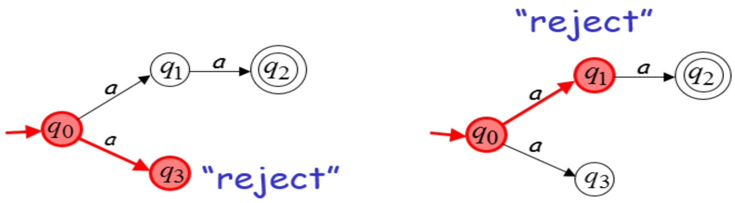
For each computation:

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

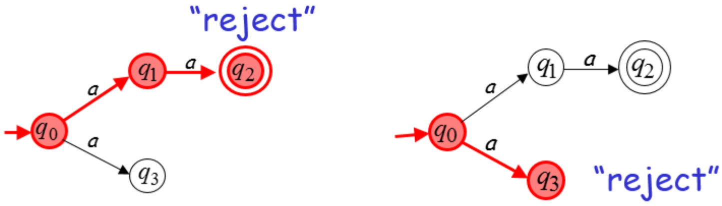
OR

- The input cannot be consumed

a is rejected by the NFA:

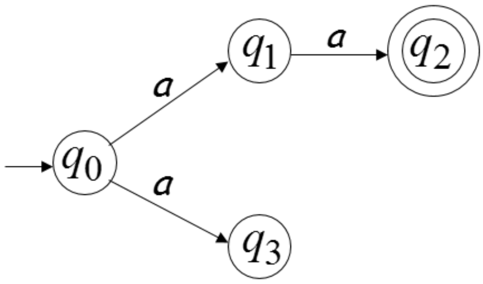


All possible computations lead to rejection aaa is rejected by the NFA:

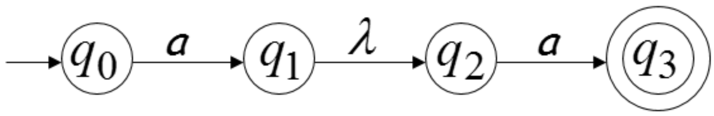


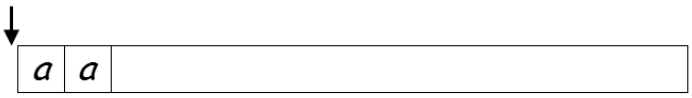
All possible computations lead to rejection

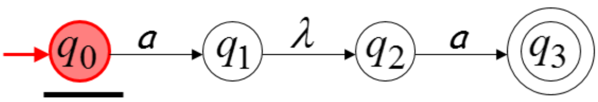
Language accepted: $L = \{aa\}$

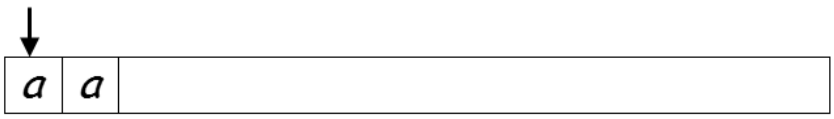


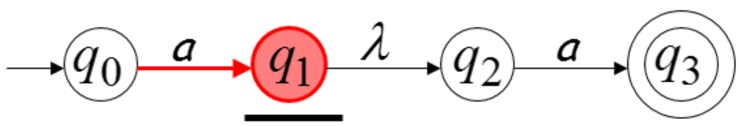
Lambda Transitions



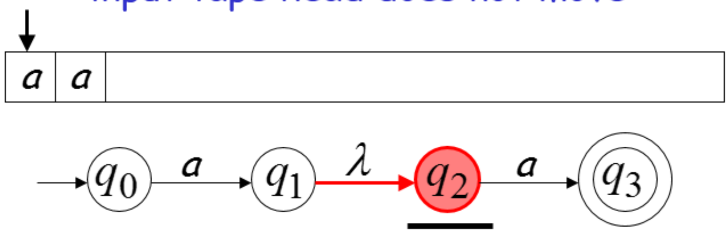






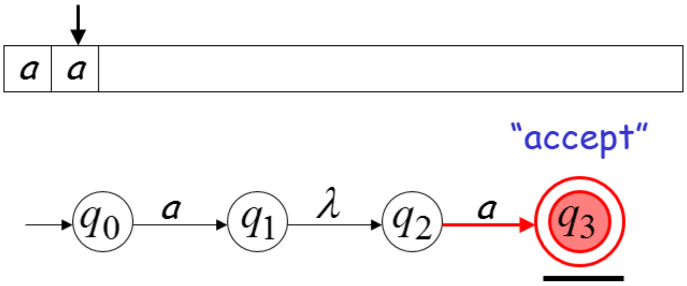


input tape head does not move



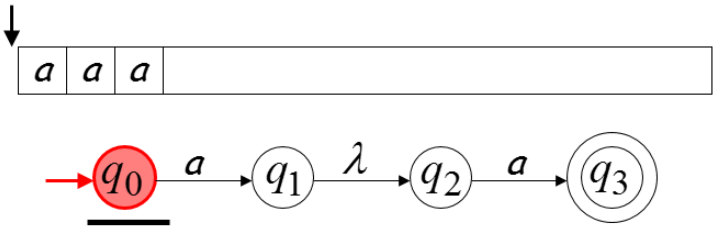
Automaton changes state

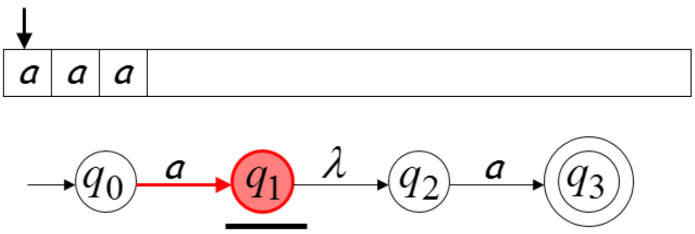
all input is consumed



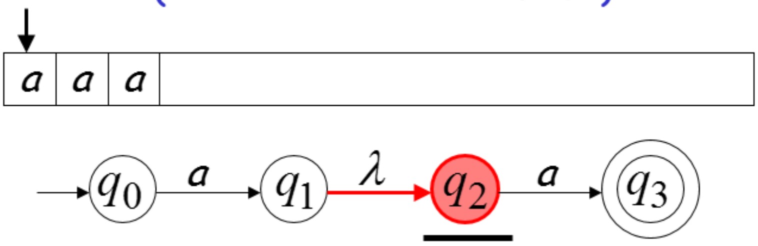
String aa is accepted

Rejection Example

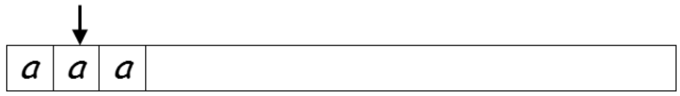




(read head doesn’t move)

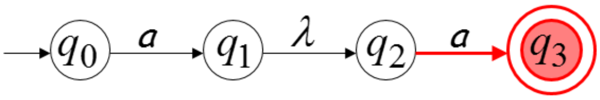


Input cannot be consumed



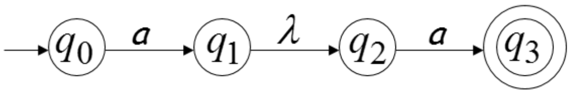
Automaton halts

"reject"

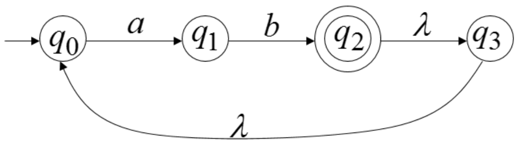


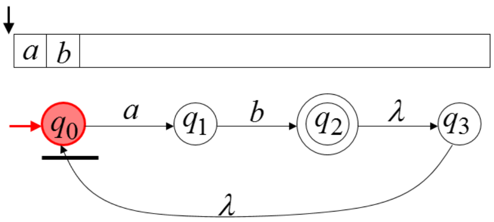
String **aaa** is rejected

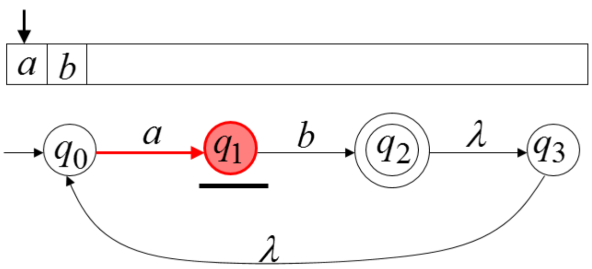
Language accepted: $L = \{aa\}$

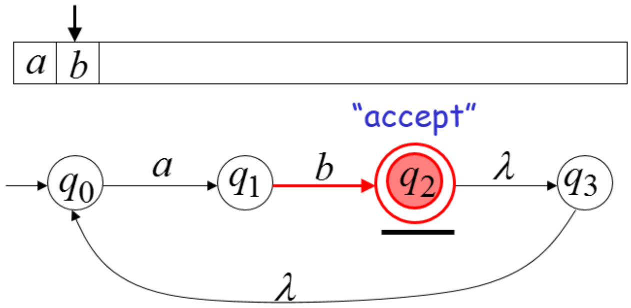


Another NFA Example

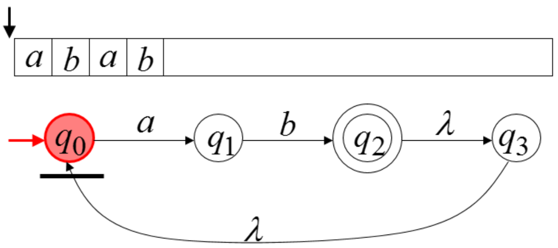


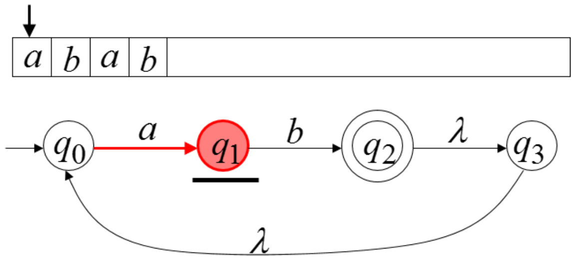


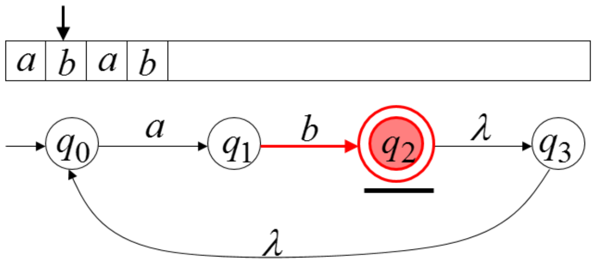


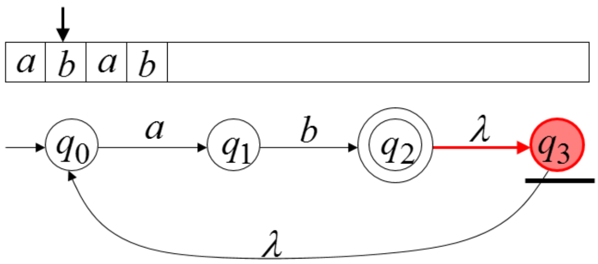


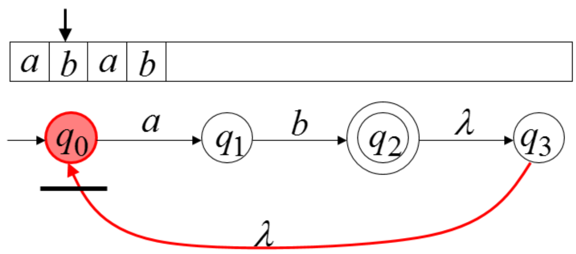
Another String

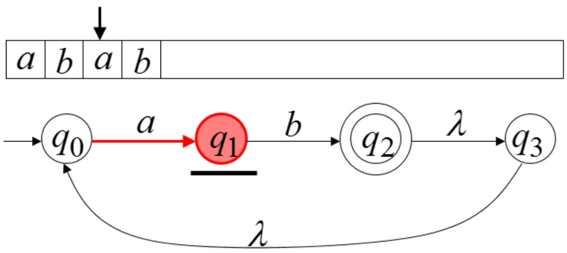


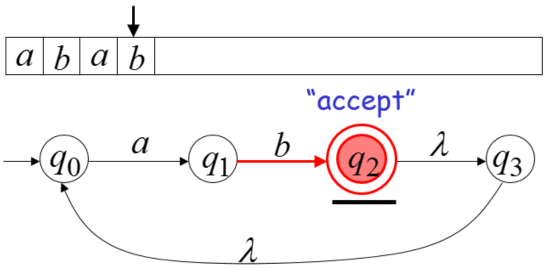






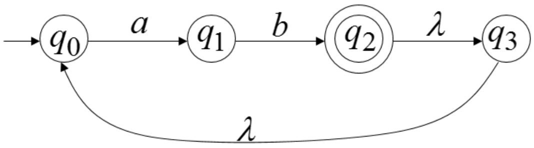




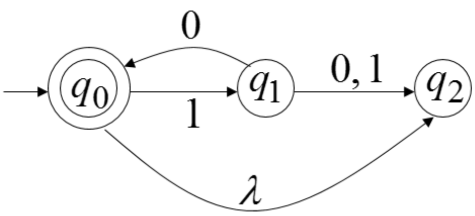


Language accepted

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

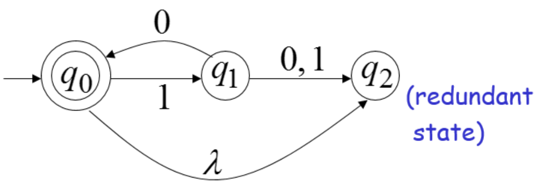


Another NFA Example



Language accepted

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

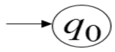


Remarks:

- The λ symbol never appears on the input tape

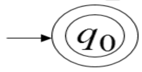
- Simple automata:

M_1



$L(M_1) = \{\}$

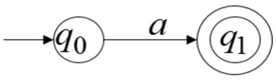
M_2



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

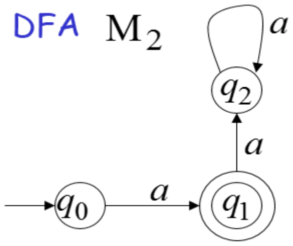
- NFAs are interesting because we can express languages easier than DFAs

NFA M_1



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

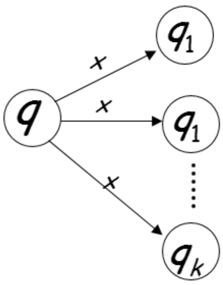
DFA M_2



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

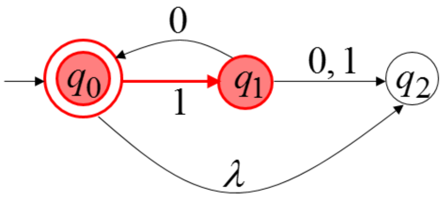
Transition Function δ

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

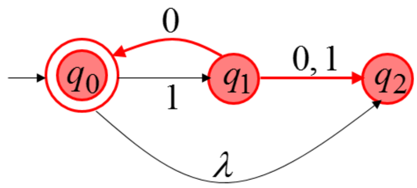


resulting states with following one transition with symbol x

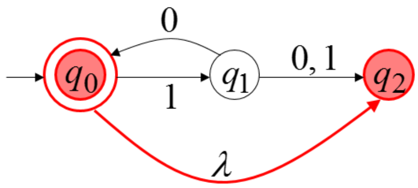
$$\delta(q_0, 1) = \{q_1\}$$



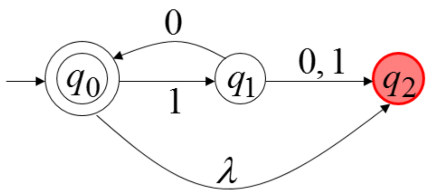
$$\delta(q_1, 0) = \{q_0, q_2\}$$



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



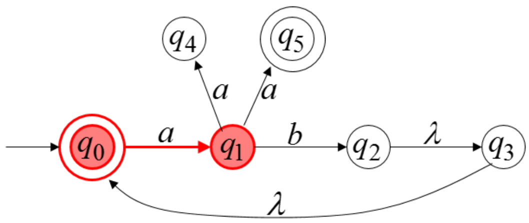
$$\delta(q_2, 1) = \varnothing$$



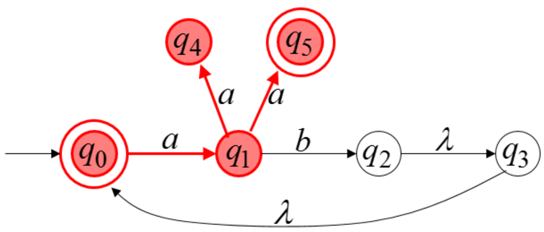
Extended Transition Function δ^*

Same with δ but applied on strings

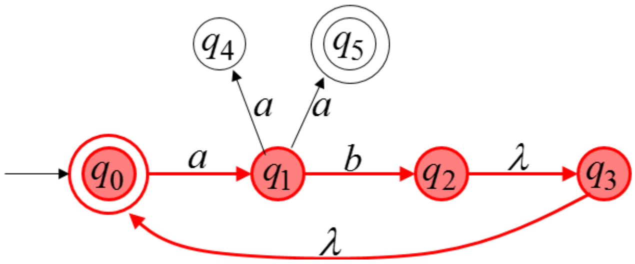
$$\delta^*(q_0, a) = \{q_1\}$$



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



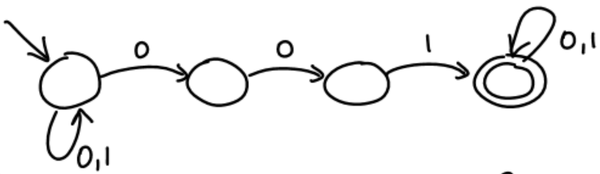
$$\delta^*(q_0, ab) = \{q_2, q_3, q_0\}$$

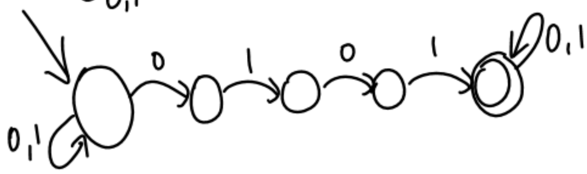


Example:

L={ w $\in$ {0,1}* | w contains 001 or 0101 as a substring }

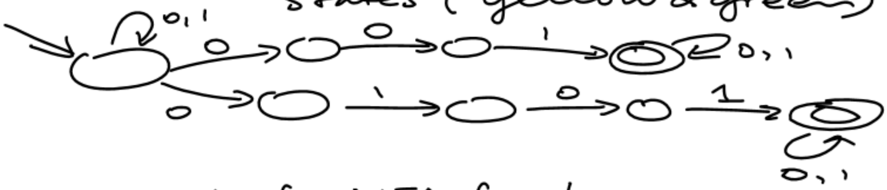
L_{001} = { w $\in$ {0,1}* | w contains 001 as a substring }





L_{0101} = { w $\in$ {0,1}* | w contains 0101 as a substring }

idea for an NFA for L:
merge the two starting states (yellow & green)



an example of a NFA for L_{001}

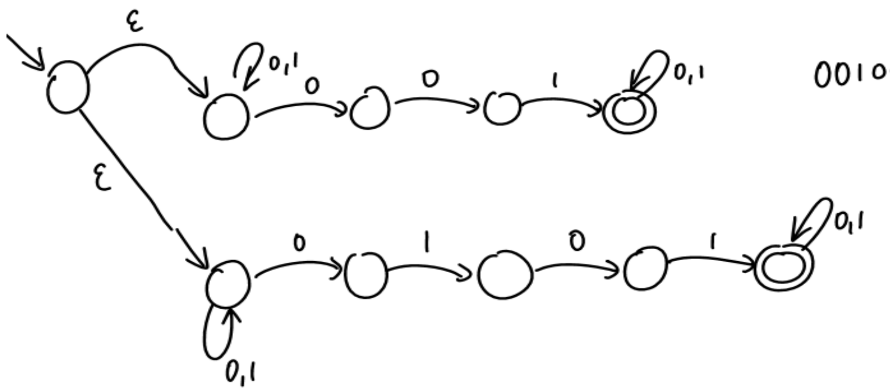
e.g. input: 1101000111001110
is accepted
(two possible accepting computation paths)

01011110101111

Example:

{ w ∈ {0,1}* | w contains 001 or 0101 as a substring }

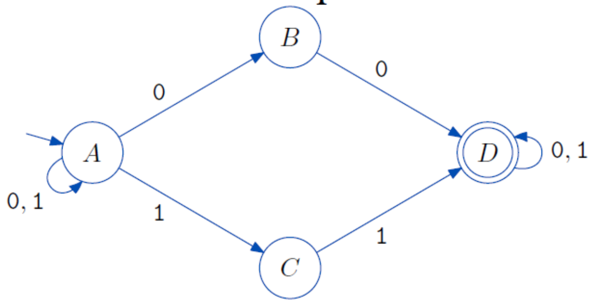
Nondeterministic FA can also use ε-transitions:



0010000101

Example: Doubles

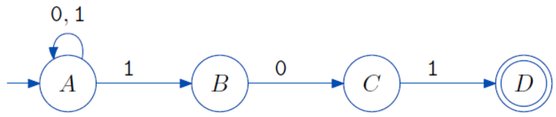
What does this NFA accept?



It accepts any binary string that contains `00` or `11` as a substring.

Example: Ending of Strings

An NFA that accepts all binary strings that end with `101`.



Example: Simultaneous Patterns

An NFA for `a`*+(`ab`)*