CHAPTER 1

Finate Automata, Regular Expressions, & Regular Languages

So Far….

- Review on sets and sequences
- General overview of languages

Models of Computation

- Computability Theory
  - Deals with the mathematical basis for Computer Science
    - Help us answer the question: "What can and cannot be computed"
- Computational Model
  - Mathematical object (defined on paper) that enables us to reason about computation and to study the properties and limitations of computing.
- Can we build a simple model of computation?

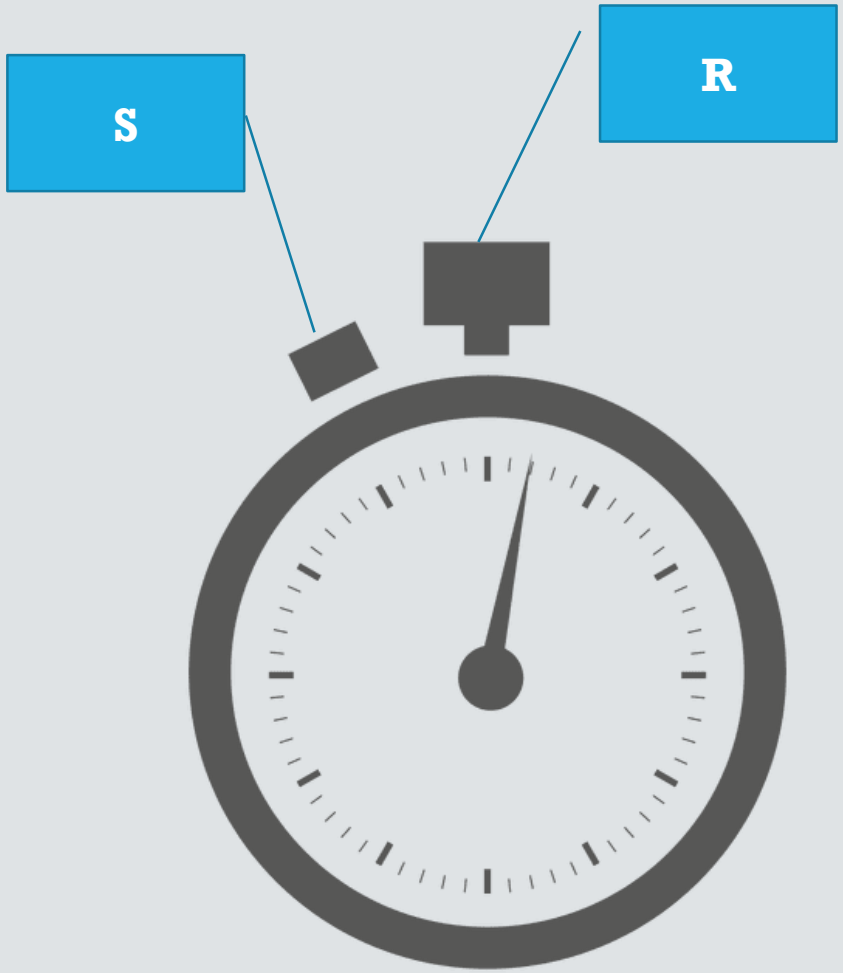
FINITE AUTOMATA

Good model for computers with an extremely limited amount of memory
Useful tools when attempting to recognize patterns

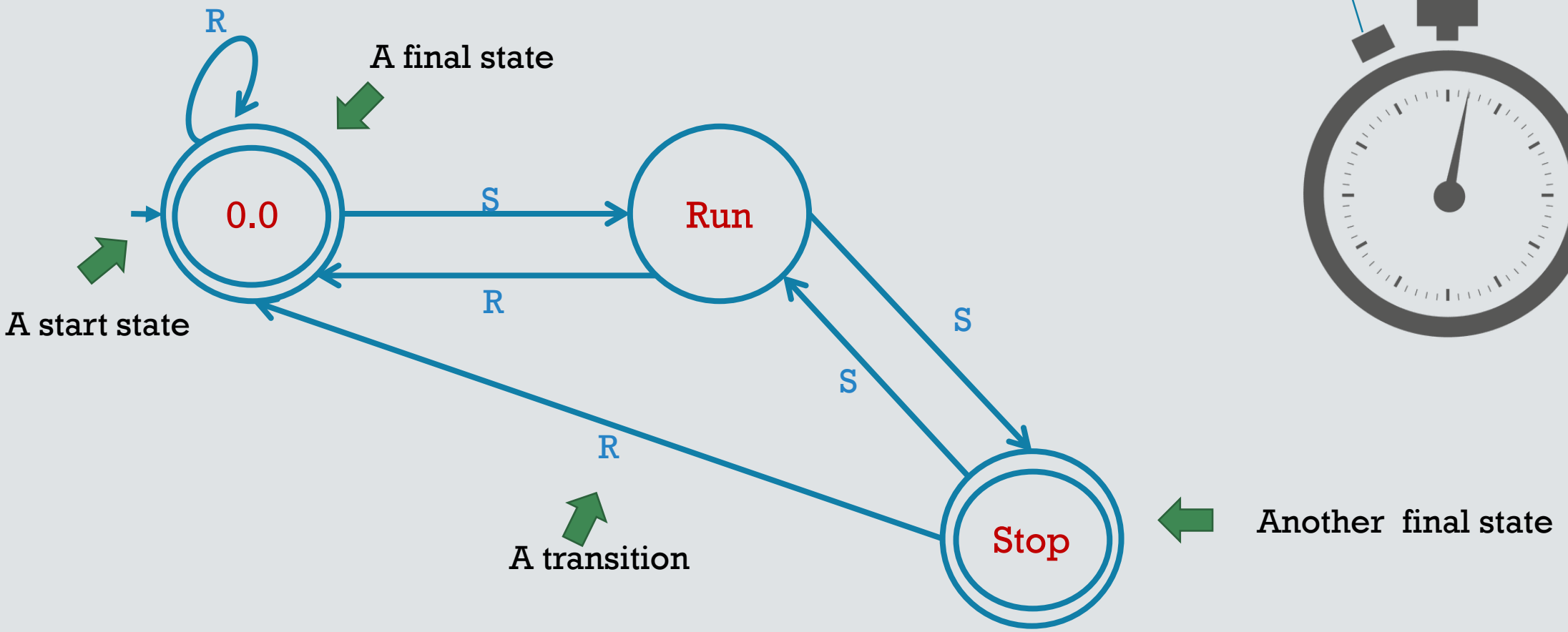
A Stopwatch

Assumptions

- The watch has 2 controllers reset (**R**) and start/stop (**S**)
- Reset sets the watch back to 0
- Start/stop take care of time-keeping
- S starts if watch is not running, otherwise it stops the watch
- To finalize the process the stopwatch must either *be at 0.0* or *not be running*



A Stopwatch


R
A final state
0.0
S
Run
R
A start state
S
S
R
A transition
Stop
Another final state

S
R

So Far…

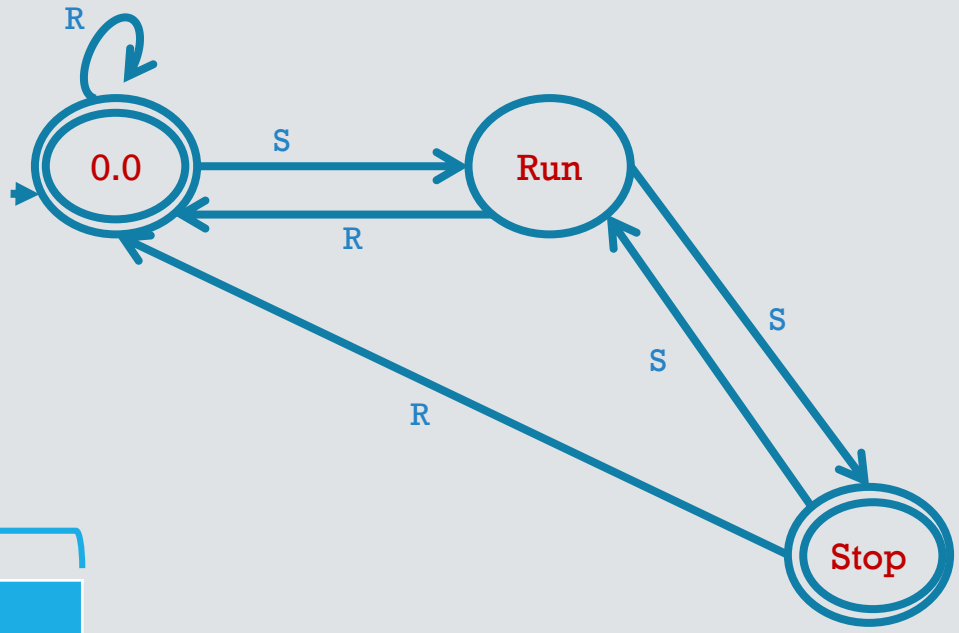
- Finite Automaton
  - Is a …. model for computers with an extremely **limited amount of memory**
  - Good for … emulating **simple/few** instructions
  - Useful when…attempting to recognize **patterns**
  - Consists of
    - States
    - Transitions
    - A single start state
    - One or more final states
    - Alphabet

Finite Automaton

- Definition
  - A finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_o, F)$, where
    - Q is a finite set called the **states**
    - Σ is a finite set called the **alphabet**
    - $\delta: Q \times \Sigma \rightarrow Q$ is the **transition function**
    - $q_o \in Q$ is the **start state**
    - $F \subseteq Q$ is the set of **accept states** (or final states)
  - Observations
    - Each state has a single transition for each symbol in the alphabet
    - There can be no accepting (final) states in the FA
    - Every FA has a computation for every finite string over an alphabet
      - Given an input, the output is **accept**, if the machine ends on a final state, or **reject**, otherwise

Finite Automaton

- FA for the snack machine example
  - A finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_o, F)$, where
    - $Q = \{0.0, stop, run\}$
    - $\Sigma = \{R, S\}$
    - $\delta: Q \times \Sigma \rightarrow Q$ (below)
    - $q_o = 0.0$
    - $F = \{0.0, stop\}$



Alphabet symbols

States

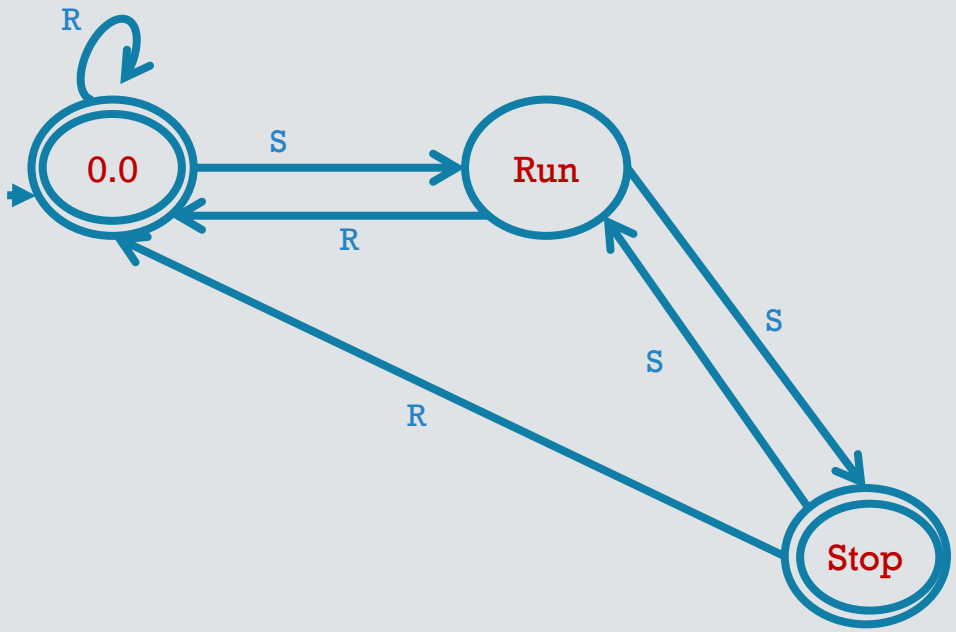
δ	R	S
0.0	0.0	Run
Run	0.0	Stop
Stop	0.0	Run

The Language of a FA

- The **language** recognized (or accepted) by a FA *M* contains all the input strings which can take *M* from the start to the final state
- For *M* and an input string *x*, if *M* stops in a final state after reading string *x* then *M* accepts *x*, or *x* is a member of the language recognized by *M*, i.e., x ∈ L(M)
- If A is the set of all the strings a machine *M* accepts, the *A* is the **language** of the machine *M,* i.e., *L(M)=A*
- Observations
  - A machine may accept several strings, but recognizes only one language
  - If the machine accepts no strings, it still recognizes one language, i.e., the empty language
  - Remember that there is a difference between the empty language and a language the accepts only the empty string

L(M) of the Stowatch

- What happens to the stopwatch after this sequence of actions: rssr?
  - Basically, what we want to know is rssr ∈ **L(M)**?



$[0.0, rssr] \vdash_M [0.0, ssr]$
$\vdash_M [Run, sr]$
$\vdash_M [Stop, r]$
$\vdash_M [0.0, \in]$

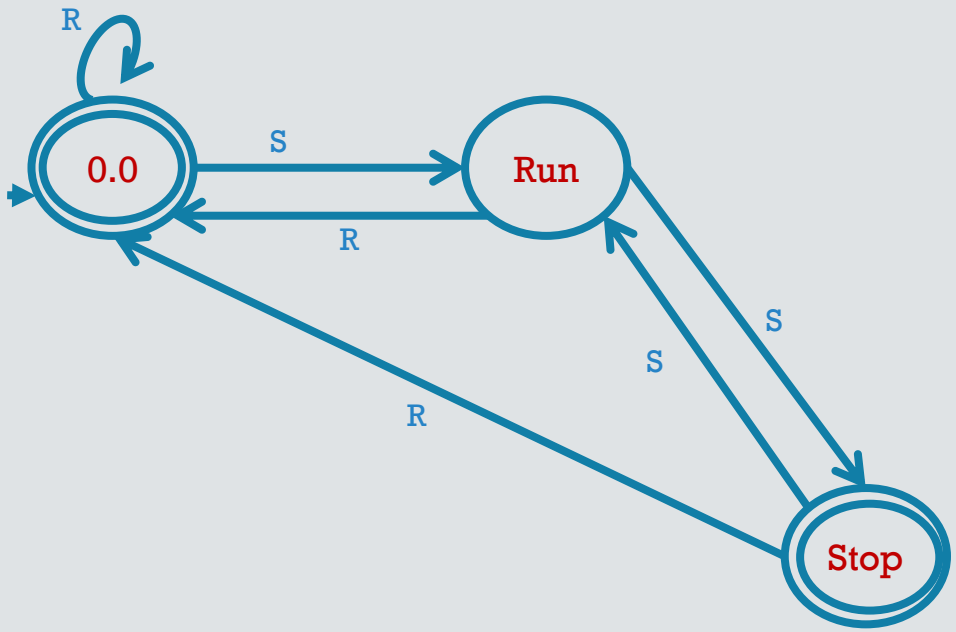
Final state

Done with string

Accept!

L(M) of the Stowatch

- What happens to the stopwatch after this sequence of actions: rs?
  - Basically, what we want to know is rs ∈ **L(M)**?



$[0.0, rs] \vdash_M [0.0, s]$

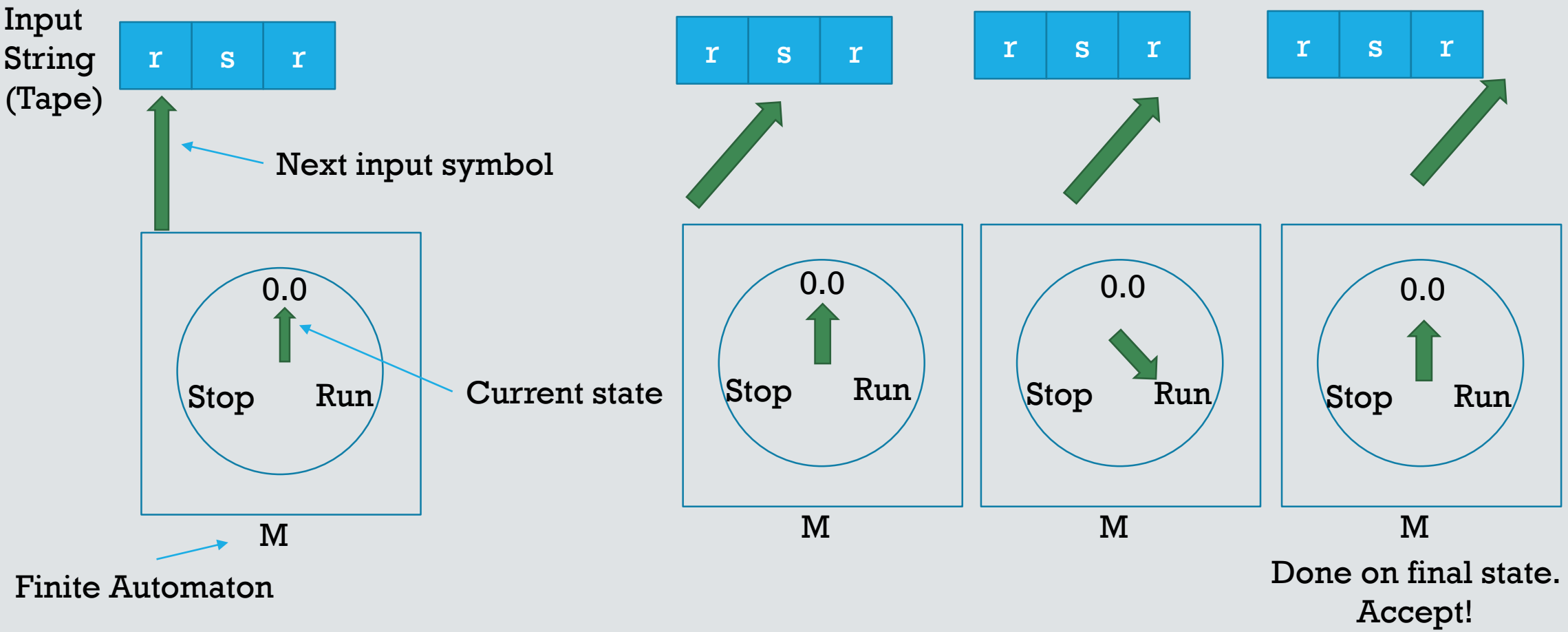
$\vdash_M [Run, \in]$

Not a final state

Done with string

Reject!

L(M) of the Stopwatch



A Snack Machine Example

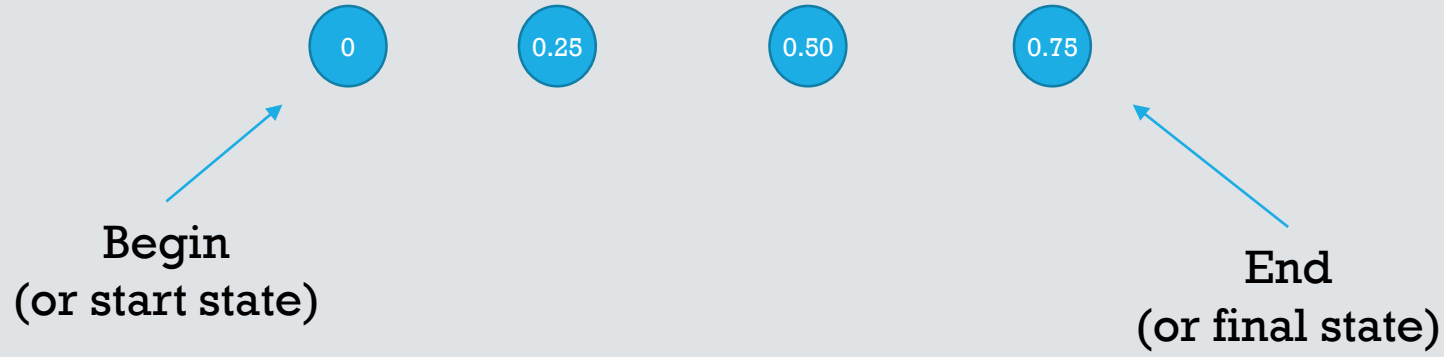
- Assumptions
  - Every item costs 75 cents
  - This machine takes
    - Quarters (q)
    - Fifty cent coins (f)
    - One dollar coins (d)
  - After 75 cents are deposited, the user can push the button "Item" (b) and the machine resets
  - The machine does not give change for inputs higher than 75 cents
  - Items never run out

A Snack Machine Example

Input	Output
Quarter	
Quarter	
Quarter	
Button	**An item**
Quarter	
Dollar	
Button	**An item**
Fifty cents	
Fifty cents	
Button	**An item**

A Snack Machine Example

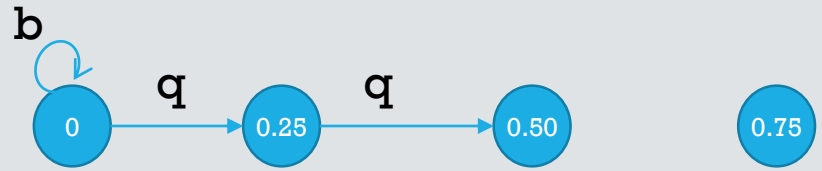
- The machine understands very few *instructions*
  - Depositing a coin
  - Pushing "Item" button
- The *state* of the machine is the amount of change deposited since the last time the machine reset



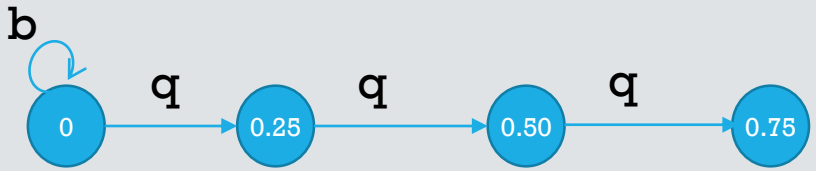
A Snack Machine Example

A Snack Machine Example

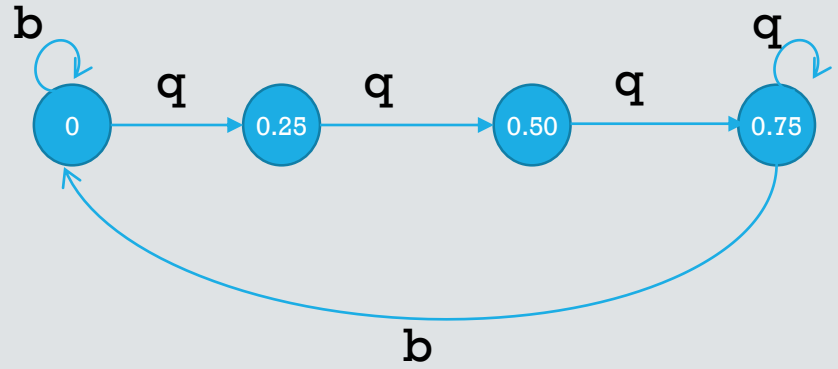
A Snack Machine Example



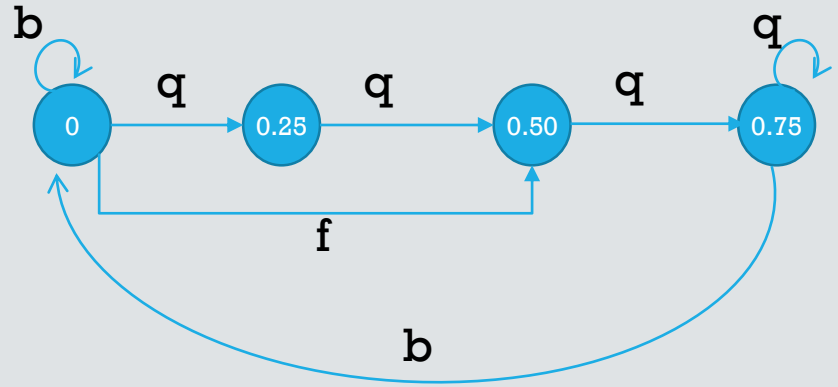
A Snack Machine Example



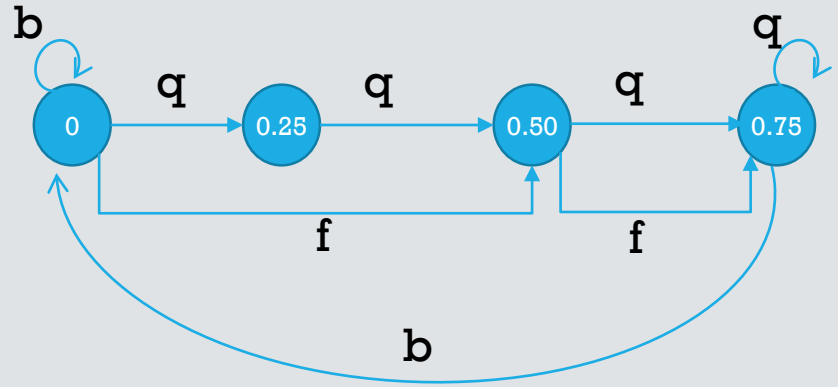
A Snack Machine Example



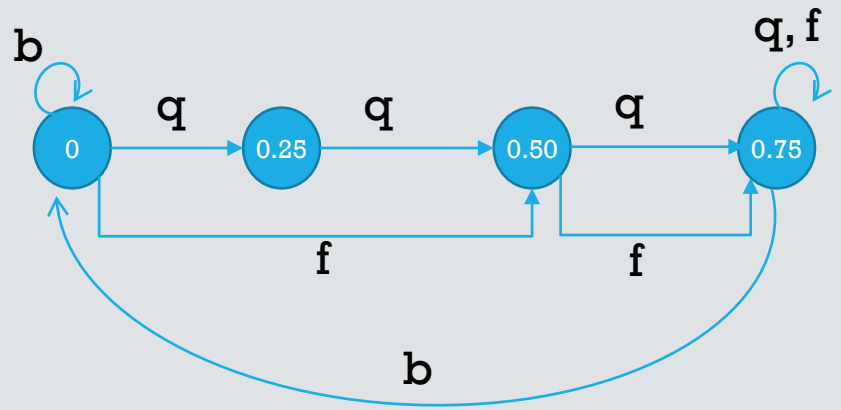
A Snack Machine Example



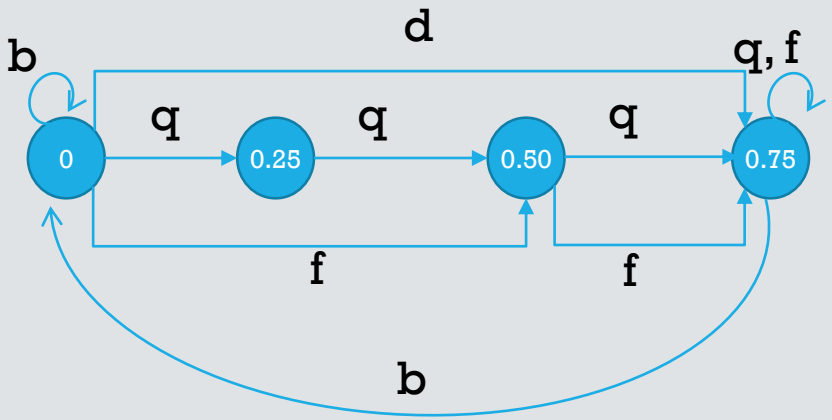
A Snack Machine Example



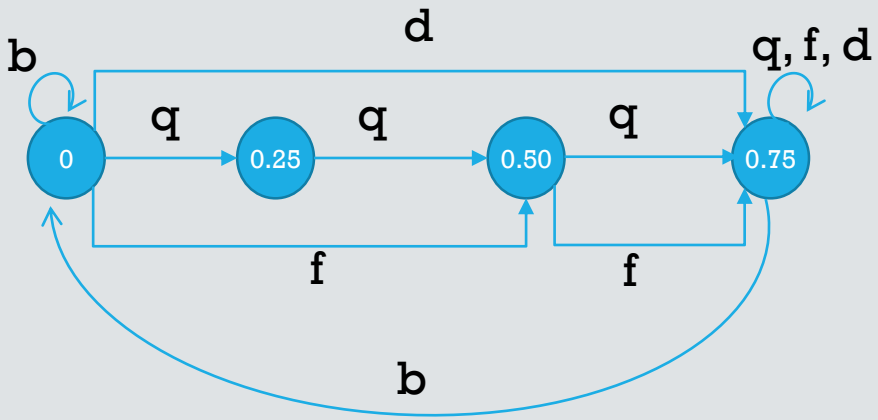
A Snack Machine Example



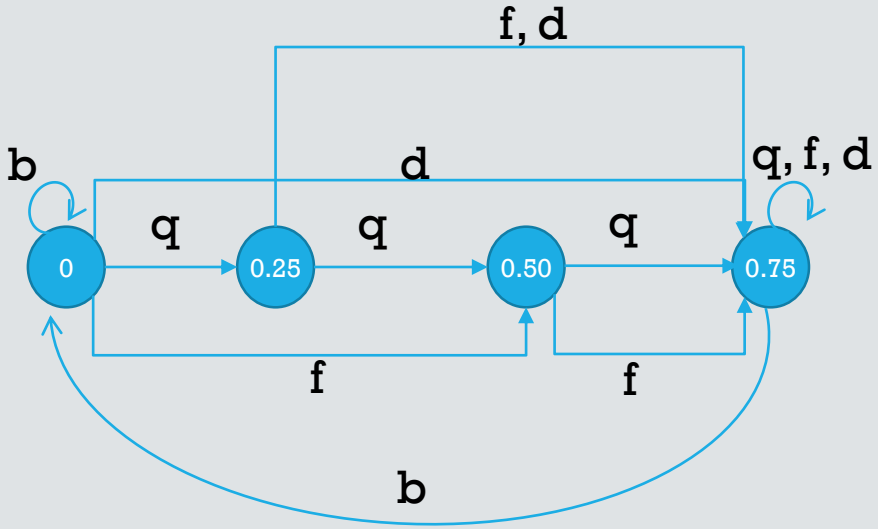
A Snack Machine Example



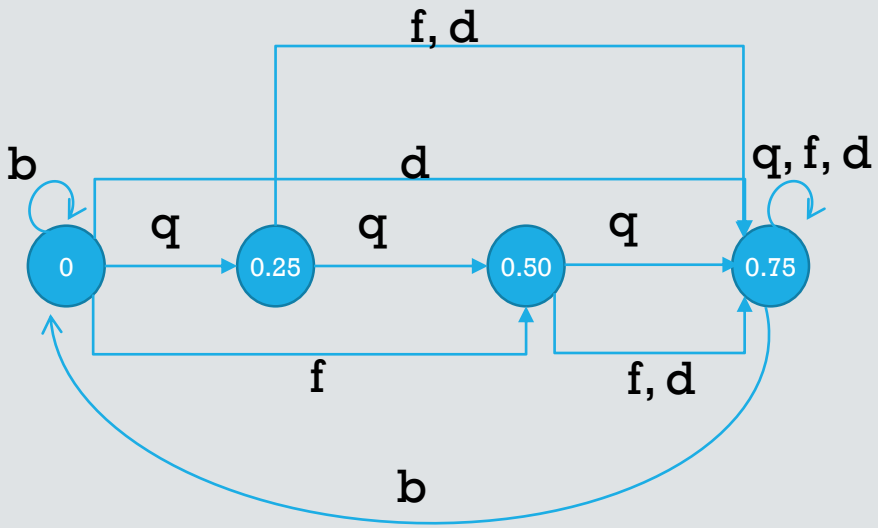
A Snack Machine Example



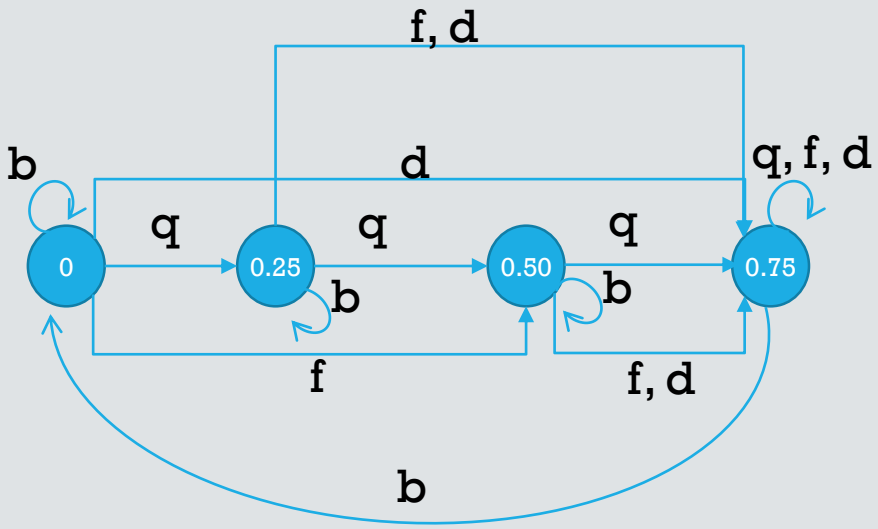
A Snack Machine Example



A Snack Machine Example

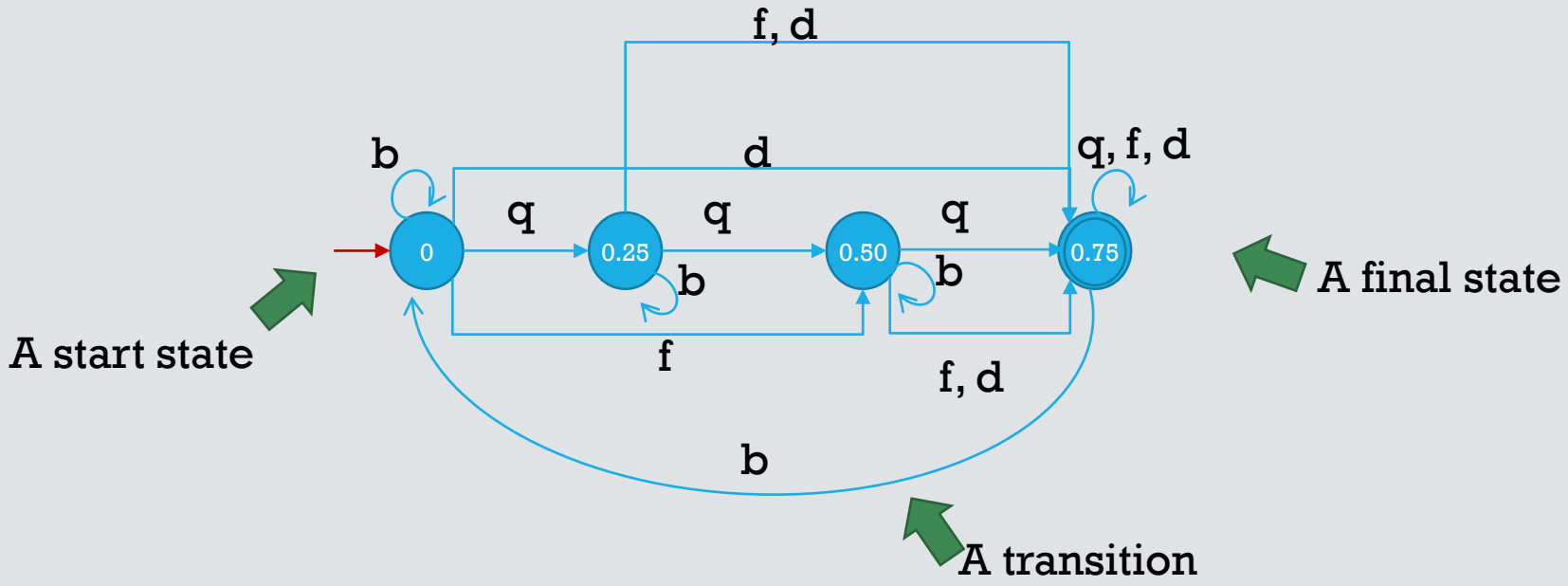


A Snack Machine Example



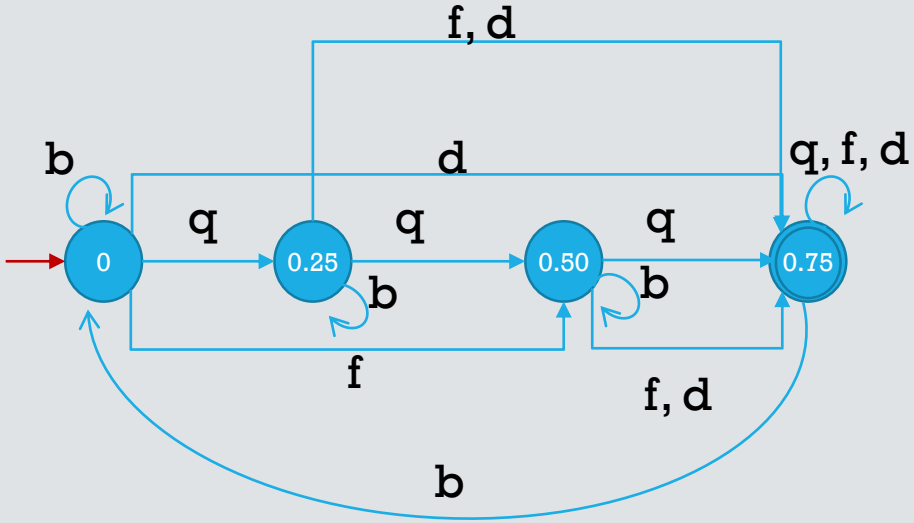
Finite Automaton

- Informally
  - FA = state transition diagram + a start state + some final states



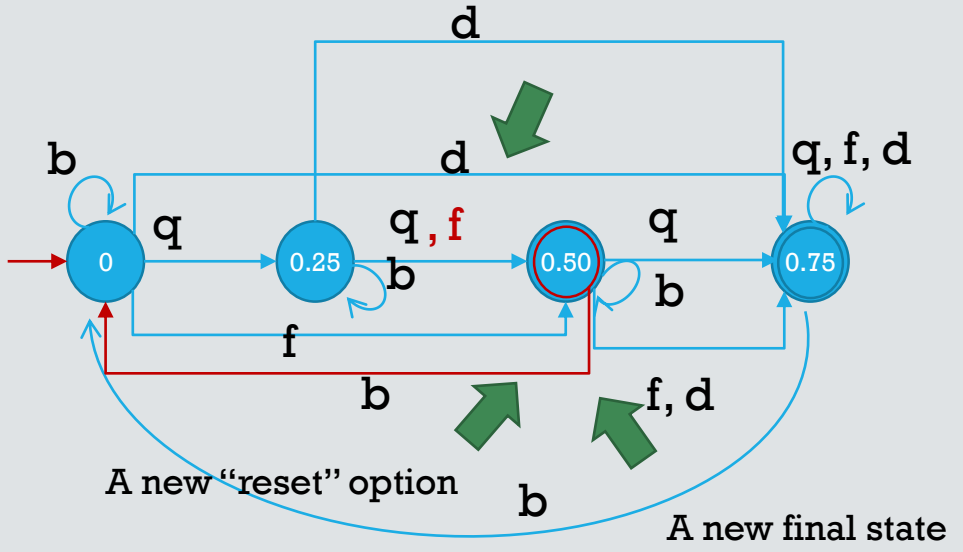
Finite Automaton

- Informally
  - FA = state transition diagram + a start state + some final states



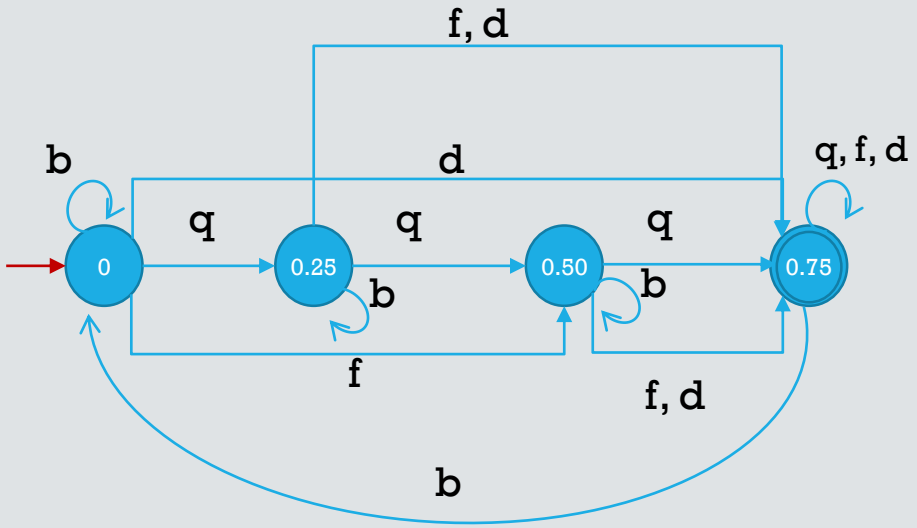
A 0.75-item machine

More "input" alternatives



A 0.5-item and 0.75-item machine

Finite Automaton



- FA for the snack machine example
  - A finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_o, F)$, where
    - Q ={0, 0.25, 0.50, 0.75}
    - Σ ={q, f, d, b}
    - $\delta: Q \times \Sigma \rightarrow Q$ (below)
    - $q_o = 0$
    - $F = \{0.75\}$

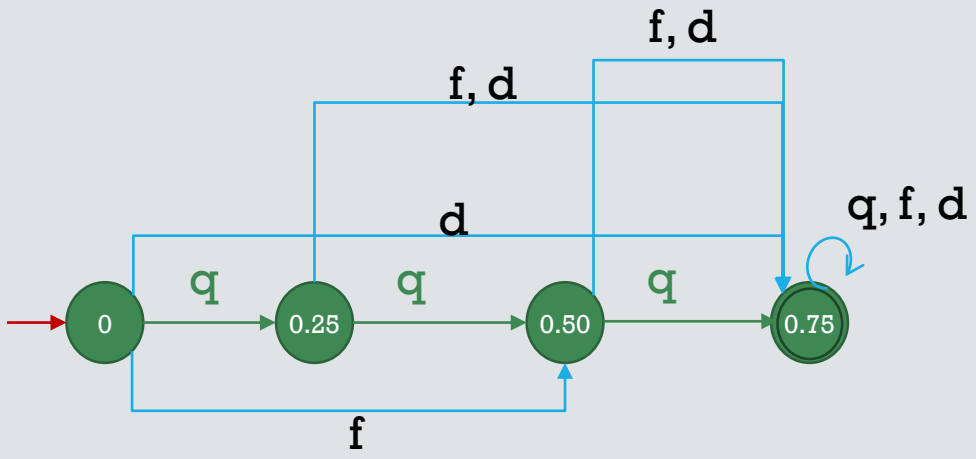
Alphabet symbols

States

δ	q	f	d	b
0	0.25	0.5	0.75	0
0.25	0.50	0.75	0.75	0.25
0.50	0.75	0.75	0.75	0.50
0.75	0.75	0.75	0.75	0

L(M) of the Snack Machine

- Will I get an item if I have the following coins qqq?
  - Basically, what we want to know is **qqq** ∈ **L(M)**?



$[0, qqq] \vdash_M [0.25, qq]$
$\vdash_M [0.50, q]$
$\vdash_M [0.75, \in]$

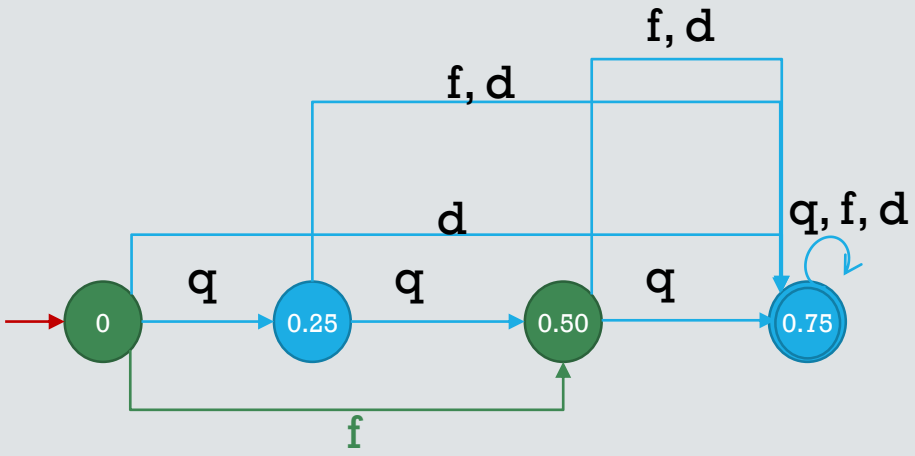
Final state

Done with string

Accept!

L(M) of the Snack Machine

- Will I get an item if I have the following coin f?
  - Basically, what we want to know is **f** ∈ **L(M)**?



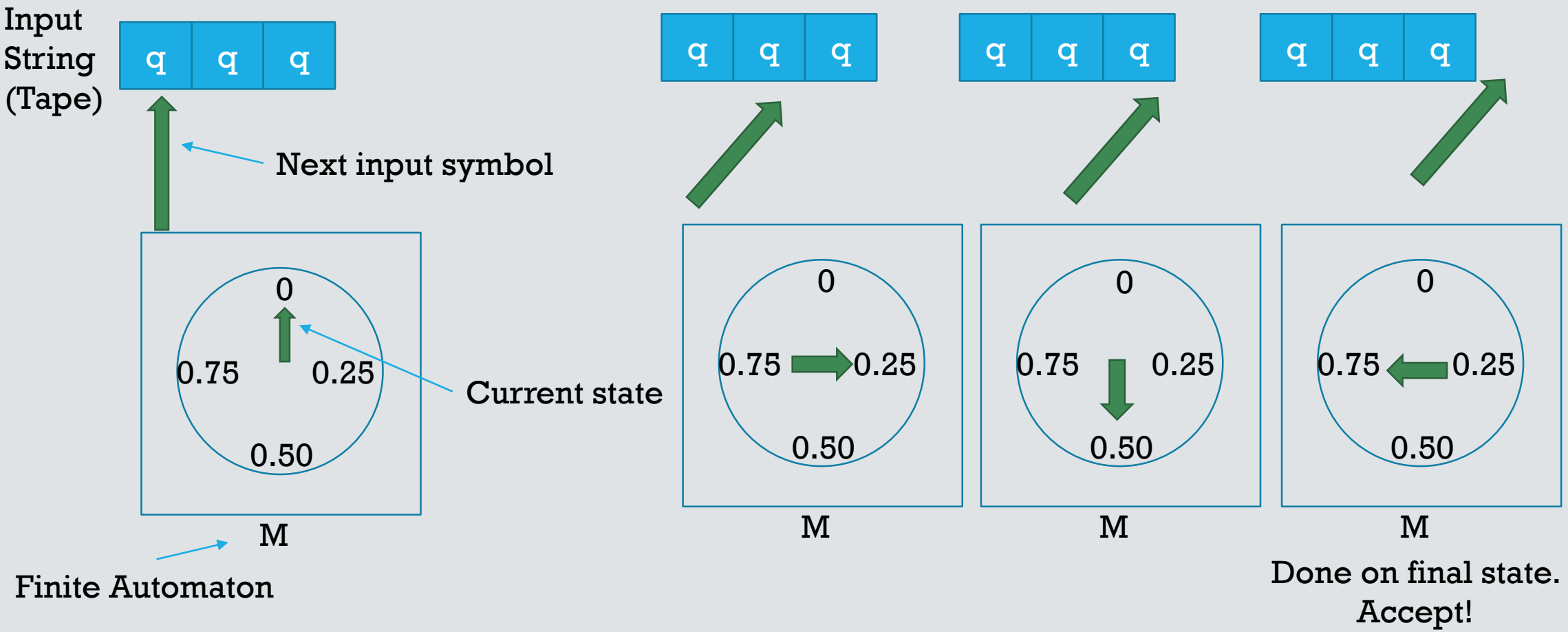
$[0, f] \vdash_M [0.50, \in]$

Done with string

Not a final state

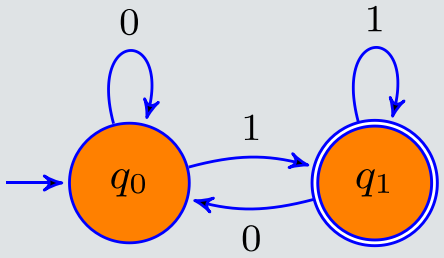
Reject!

L(M) of the Snack Machine



Example 1 – Finite Automaton

- Consider the state diagram of finite automaton M_1
  - What is the formal description of M_1 ?
  - What is the language recognized by M_1 ?



$M_1 = (Q, \Sigma, \delta, q_0, F)$ where

$Q = \{q_0, q_1\}$

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

$q_0 = q_0$

$F = \{q_1\}$

$\delta =$

	0	1
q_0	q_0	q_1
q_1	q_0	q_1

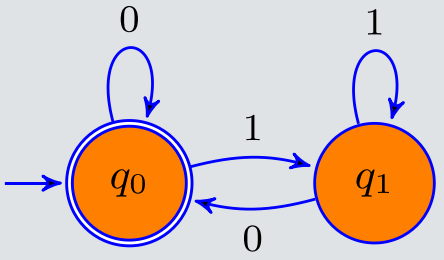
Accepted strings

- 01011
- 1
- 01101

$L(M_1) = \{w \mid w \text{ ends with a } 1\}$

Example 2 - Finite Automaton

- Consider the state diagram of finite automaton M_2
  - What is the formal description of M_2 ?
  - What is the language recognized by M_2 ?



$$M_2 = (Q, \Sigma, \delta, q_0, F) \text{ where}$$

$Q = \{q_0, q_1\}$

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

$q_0 = q_0$

$F = \{q_0\}$

$\delta =$

	0	1
q_0	q_0	q_1
q_1	q_0	q_1

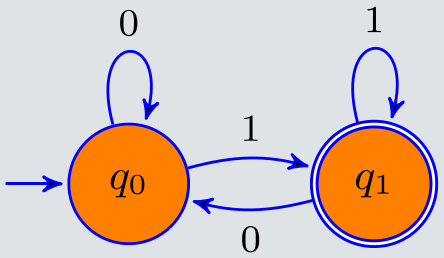
Accepted strings

- 0
- 01011100
- 0110
- ϵ

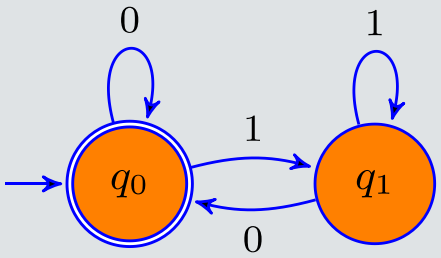
$L(M_2)$ = {w | w ends with a 0 or is the empty string}

Complementing Languages

- By swapping final states and non-final states of M_1, we get a new finite automaton M_2, such that $L(M_2) = \overline{L(M_1)}$, i.e., the complement of $L(M_1)$.



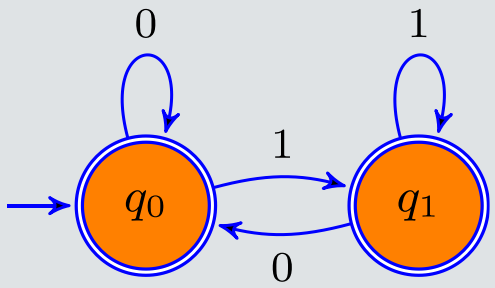
M_1



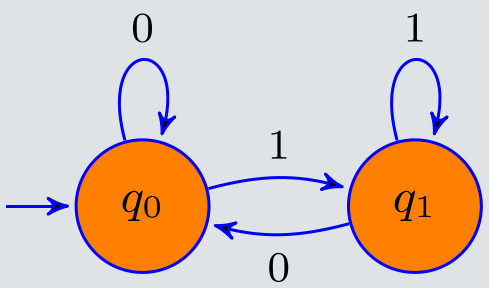
M_2

Examples of the Machines (1)

- What are the languages of these automata?



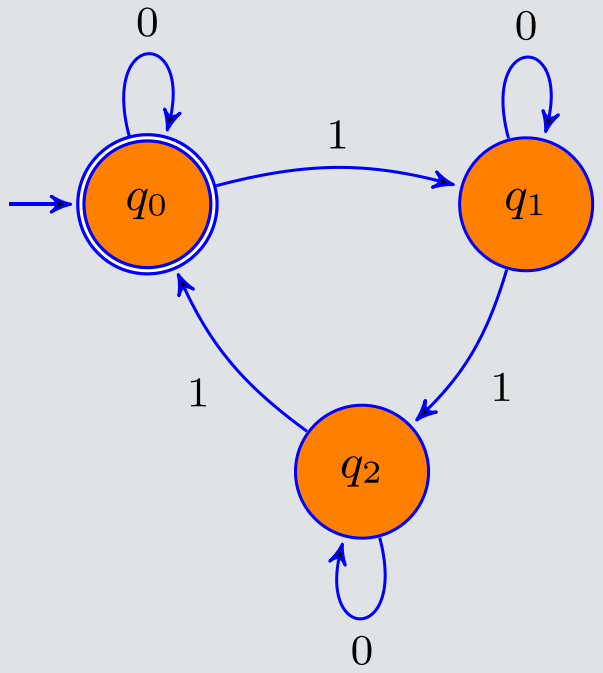
M_3



M_4

In-Class Exercise

- Consider this automaton.
- Write 5 strings that it accepts.
- Write 5 strings that it rejects.
- What is the language of M_6?



M_6

Regular Languages

- Remember that a language is a set of strings over some alphabet
- M recognizes a language A if $A = \{w \mid M \text{ accepts } w\}$
- Definition
  - A language is called a ***regular language*** if some finite automaton recognizes it

How to Design a DFA?

- Given a regular language L, how do we construct an automaton M that recognizes that language?
- Three steps to design a finite automaton M to recognize a language L.
  1. Determine the information to remember about the part of an input string which M has already read.
  2. Represent that information as a set of states. The start state is the state corresponding to the empty string. A final state is a state corresponding to string in L.
  3. Assign a transition for each input symbol from each state.
- Notes
  1. If the information remembered in step 1 is insufficient, step 3 will have problems
  2. The information needed to remember in step 1 my not be unique.
- Let's try: Design a finite automaton M_1 that recognizes the following language:

$$L_1 = \{x \text{ over } \{0,1\} \mid x \text{ does not end with } 01\}$$

In-Class Exercises

- **Design a finite automaton M_2 that recognizes the following language**

$$L_2 = \{x \text{ over } \{0,1\} \mid x \text{ contains substring } 10\}$$

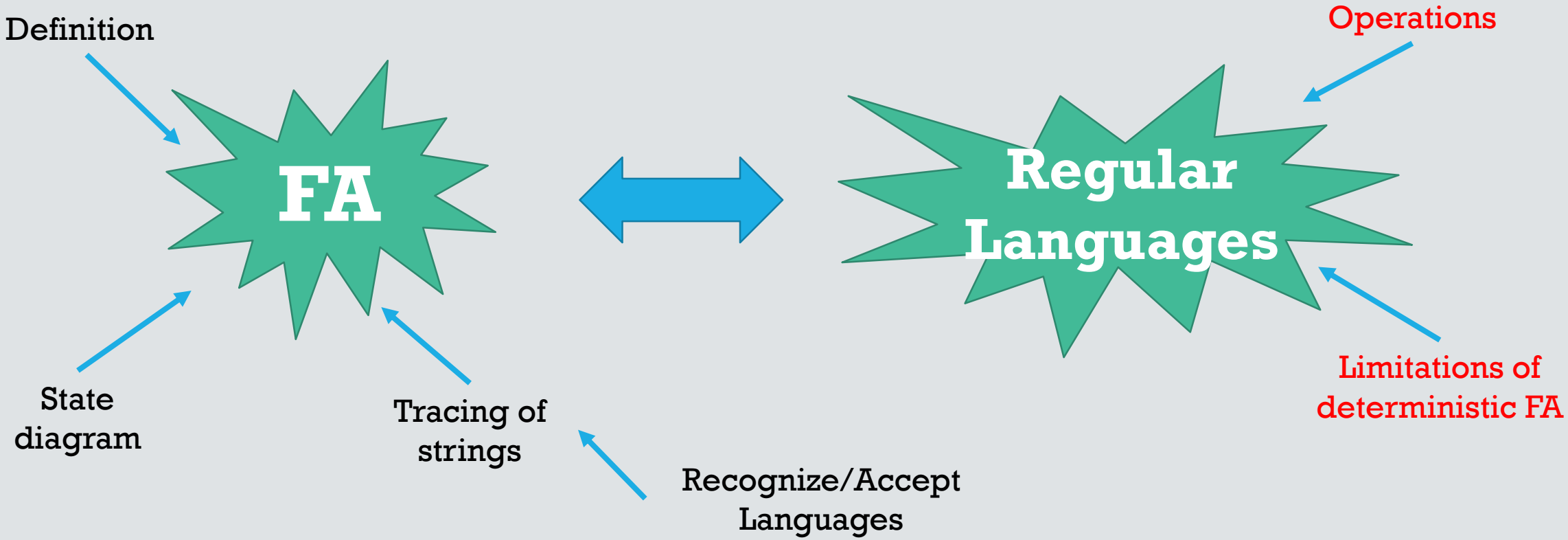
- **Design a finite automaton M3 that recognizes the following language**

$$L_3 = \{x \text{ over } \{0,1\} \mid x \text{ does not end with } 01 \text{ and contains substring } 10\}$$

$$L_3 = L_1 \cap L_2$$

  - **We will come back to L_3 again later.**

So Far…


Definition
FA
Operations
Regular Languages
State diagram
Tracing of strings
Limitations of deterministic FA
Recognize/Accept Languages

Operations

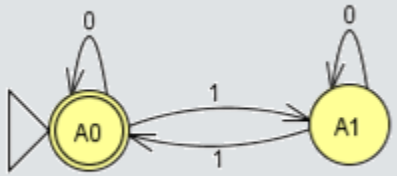
- Definition
  - Let A and B be languages. We define the regular operations union, concatenation, and star:
    - Union: $A \cup B = \{x | x \in A \text{ or } x \in B\}$
    - Concatenation: $A \circ B = \{xy | x \in A \text{ and } y \in B\}$
    - Intersection: $A \cap B = \{x | x \in A \text{ and } x \in B\}$
    - Star: $A^* = \{x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and } x_i \in A\}$
      - Note that the empty string ε is always a member of A*

Binary operations

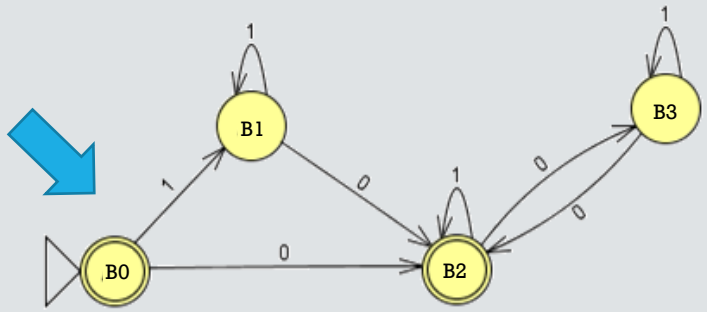
Unary operations

  - Example
    - Let A={good, bad}, B={blue, green} , and Σ ={a,b,…z}
      - $A \cup B$={good, bad, blue, green}
      - $A \cap B = \{\}$
      - $A \circ B$={goodblue, goodgreen, badblue, badgreen}
      - A*={ε , good, bad, goodgood, badbad, goodbad, badgood, badbadbad,…}

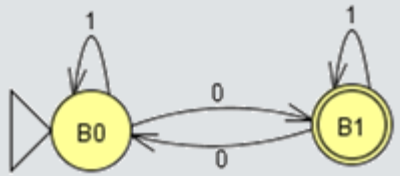
Operations & Machines



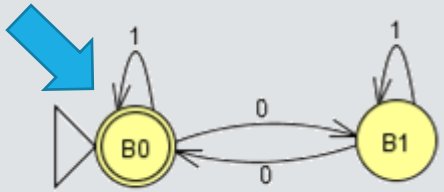
L_A= {w | w is a string with an **even** number of 1's}



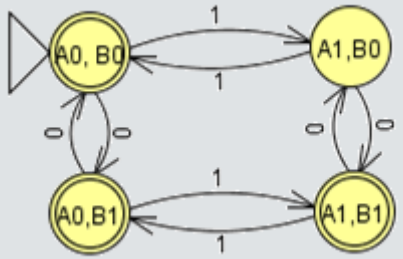
Star of L_B



L_B= {w | w is a string with an **odd** number of 0's}



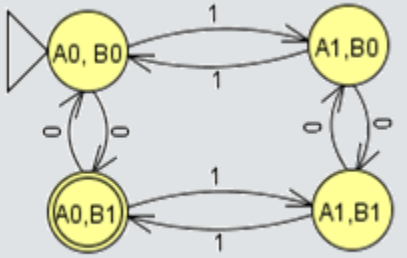
Complement of L_B



Final if either state is final on L_A **or** L_B

Union

Cross product



Final if states are final on L_A **and** L_B

Intersection

Machines for the Resulting Languages

- Let's L_1 and L_2 be regular languages
  - Each has a DFA (M_1 and M_2, respectively) that recognizes them

Can we construct M_3 using M_1 and M_2 ?

Not that simple with DFA
(Needs redesign most of the time)

Solution 1
Complete redesign

Solution 2
Intersection of L_1 and L_2

Solution NFA
Non-deterministic Finite Automaton

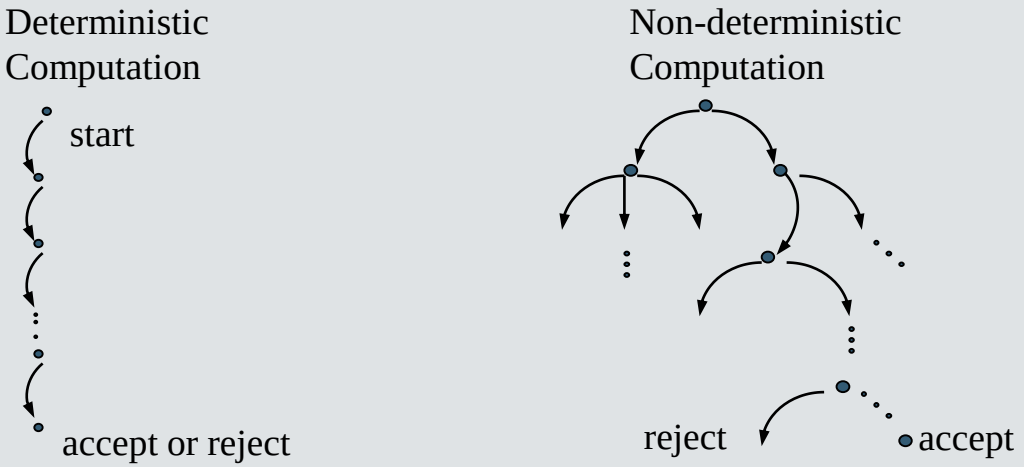
A DFA cannot figure out where to break a string for a concatenation or how to determine on which machine run the simulation for union of two languages

Lets Check In….

- How do we define a FA?
- What makes an FA deterministic?
- When tracing a string, what are the conditions to accept/reject?
- What type of languages are associated with FAs?
- Which 4 operations are closed for the type of languages above? ;)

Nondeterministic FA (NFA)

- Nondeterminism is a generalization of determinism, so every deterministic FA is a NFA vice versa
- Deterministic versus nondeterministic computations

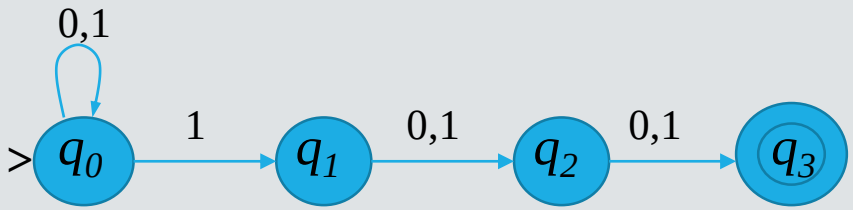


Nondeterministic FA (NFA)

- Definition
  - A nondeterministic finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_o, F)$, where
    - Q is a finite set of states
    - Σ is a finite alphabet
    - $q_o \in Q$ is the start state
    - $F \subseteq Q$ is the set of accept states
    - $\boldsymbol{\delta = Q \times \Sigma_\varepsilon \rightarrow P(Q)}$ is the transition function
      - Recall that $P(Q)$ is the power set of the set of states Q and Σ_ε is the alphabet augmented to contain the empty string ε, i.e., $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$.
- NFA versus deterministic FA
  - Relaxes the restriction that all the outgoing arcs of a state must be labeled with *distinct symbols* as in deterministic FAs
  - The transition to be executed at a *given state* can be *uncertain*, i.e., > 1 possible transitions, or no applicable transition
  - Transitions to from one state to the next without reading any input symbol are allowed

NFA - Example 1

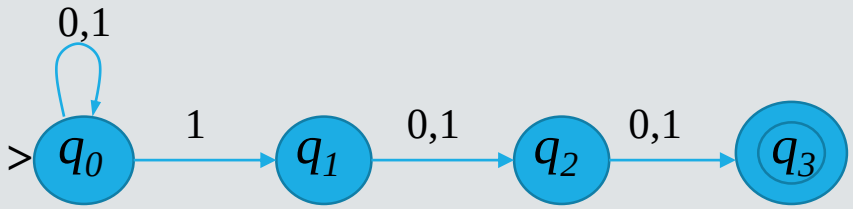
- Consider the following state diagram of NFA M_1:

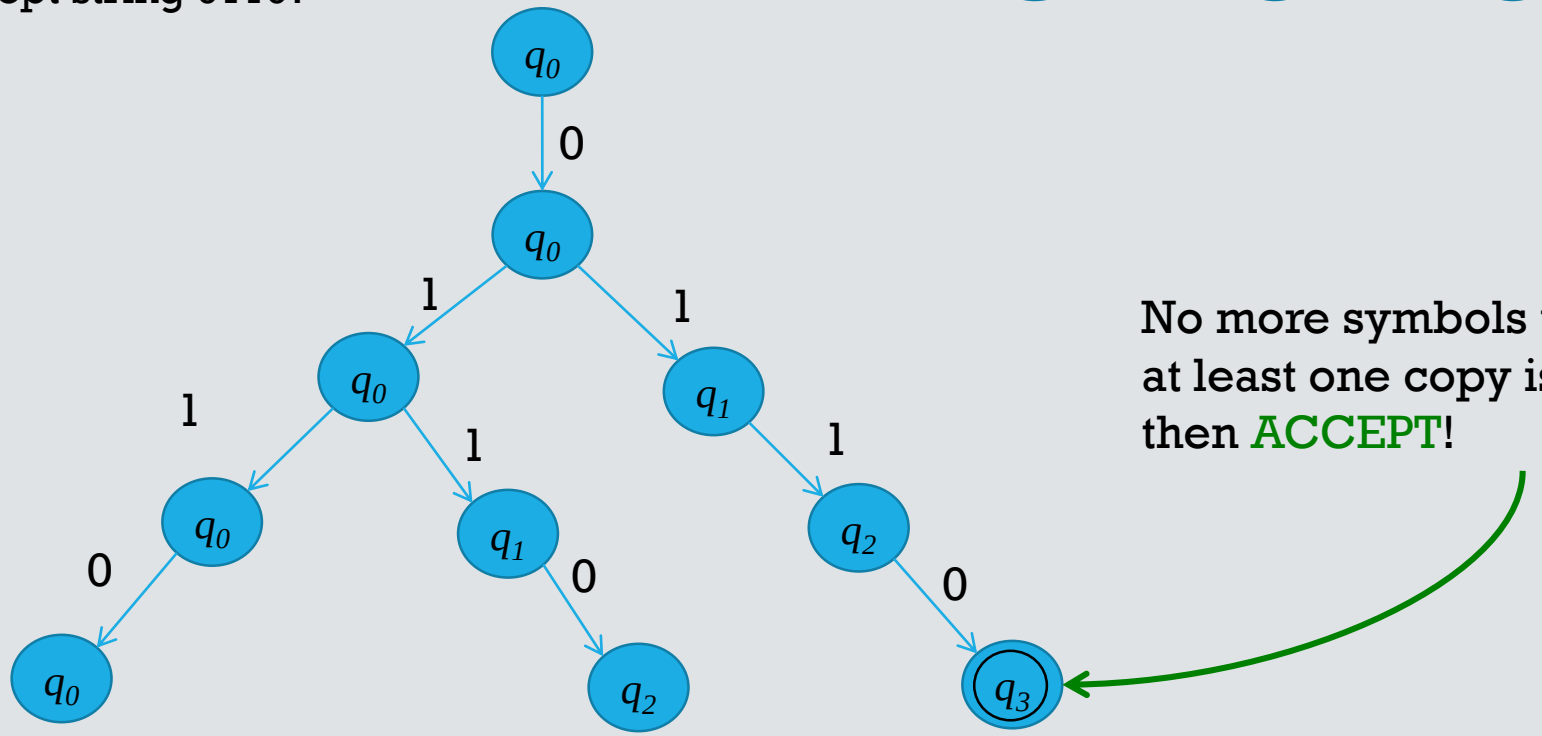


  - M_1 recognizes strings with a "1" in the third position from the end
  - M_1 stays in the start state until it "**guesses**" that it is three places from the end of the computation

NFA - Example 1

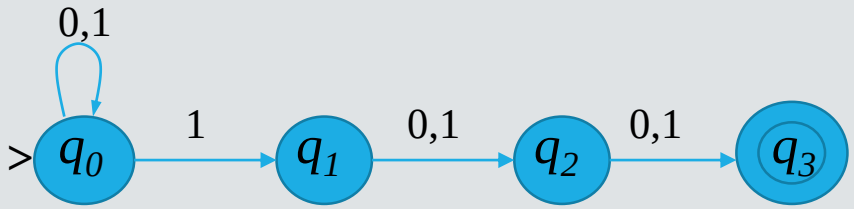
- Consider the following state diagram of NFA M_1:
  - Does it accept string 0110?

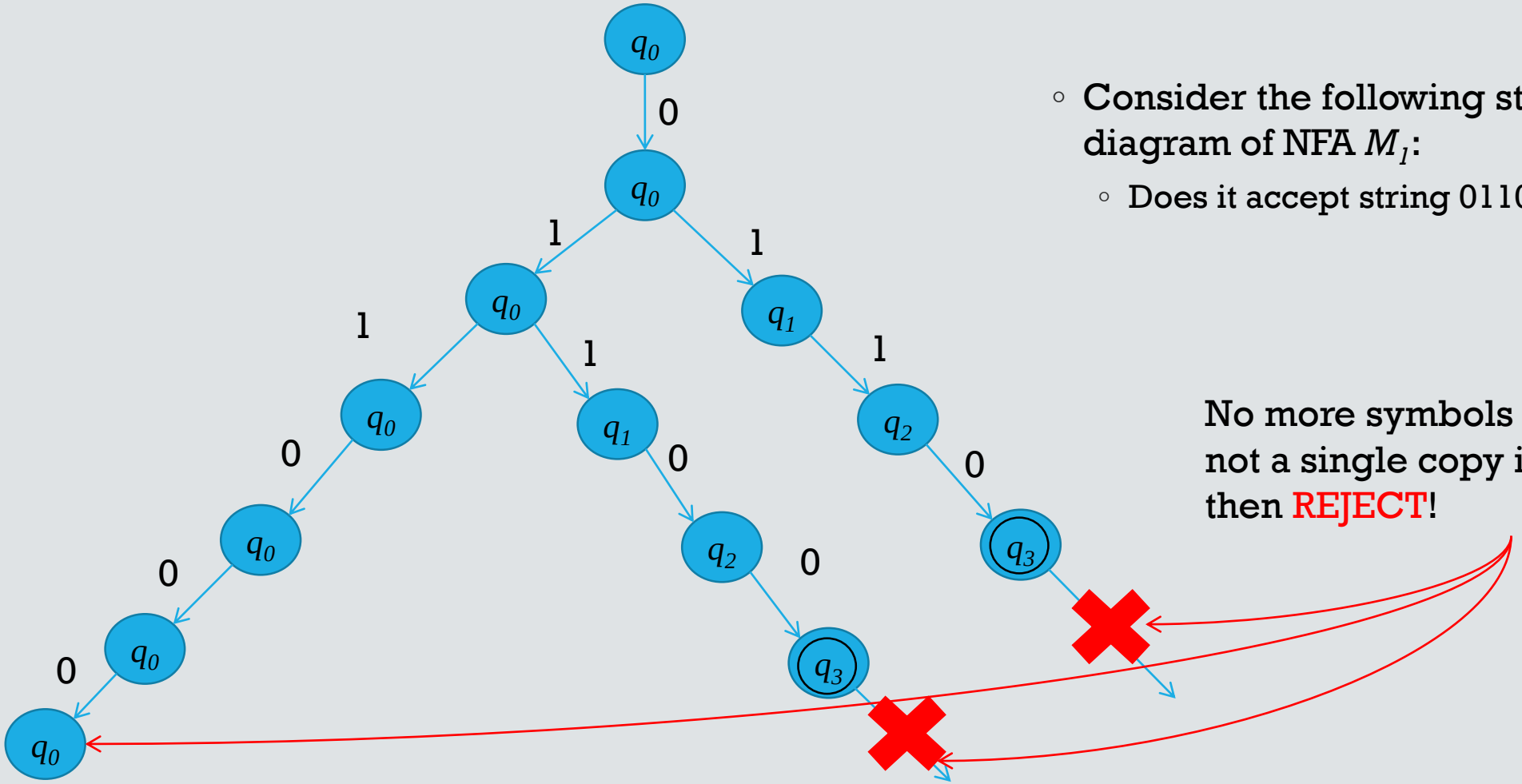




No more symbols to read and at least one copy is in a final state then ACCEPT!

NFA - Example 1



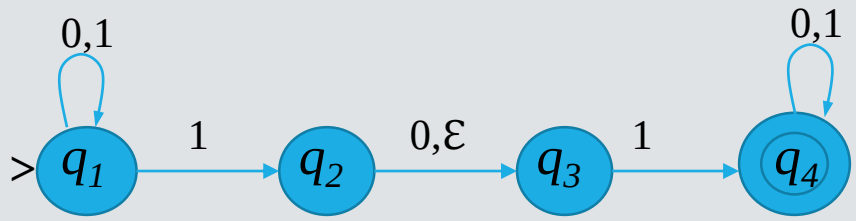


- Consider the following state diagram of NFA M_1:
  - Does it accept string 011000?

No more symbols to read and not a single copy is in a final state then REJECT!

NFA – Example 2

- Example
  - Consider the following state diagram of NFA M_2
    - What is the formal definition of M_2?
    - Which language is recognized by M_2?



$M_2 =$ $(Q, \Sigma, \delta, q_o, F)$, where

- $Q = \{q_1, q_2, q_3, q_4\}$
- $\Sigma = \{0, 1\}$
- $q_o = q_1$
- $F = \{q_4\}$
- $\delta =$

	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

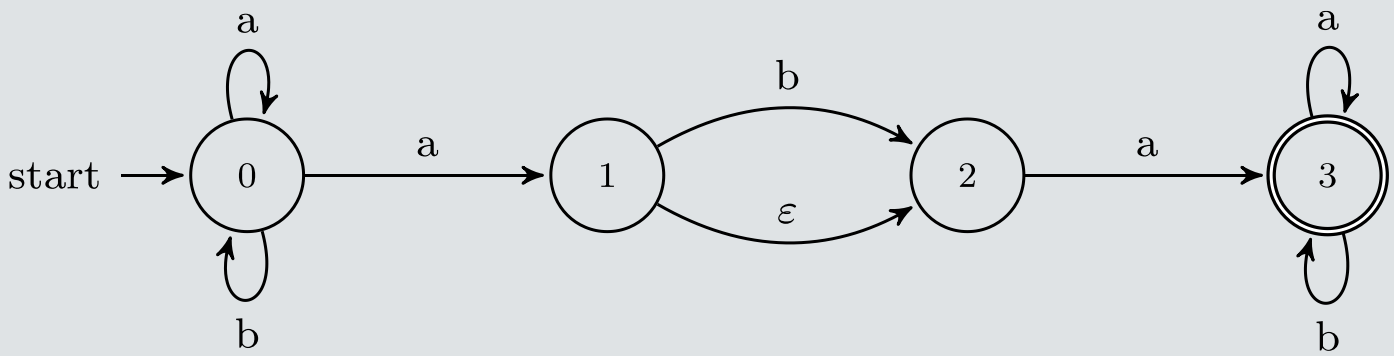
Accepted strings

- 01011
- 11
- 01101

$L(M_2) = \{w \mid w$ contains substring 101 or 11$\}$

In-Class Exercise

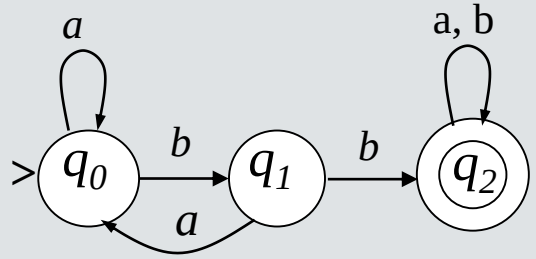
- Does this NFA accept string **baa**?



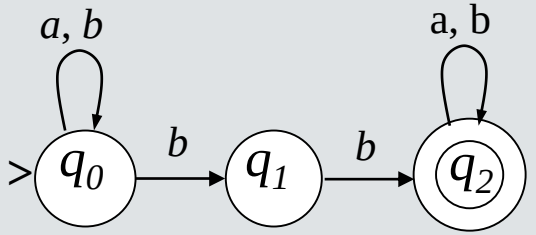
Why NFAs?

- Advantages of NFAs over DFAs
  - Sometimes DFAs have many more states, conceptually more complicated
  - Understanding the functioning of the NFAs is much easier
  - Example 1: M_1 and M_2 accept strings over {a, b} if they contain the substring "bb"

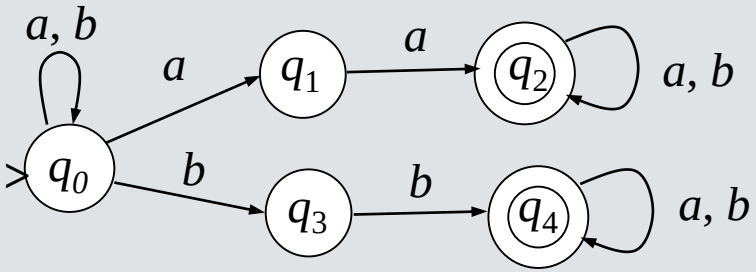
M_1:



M_2:

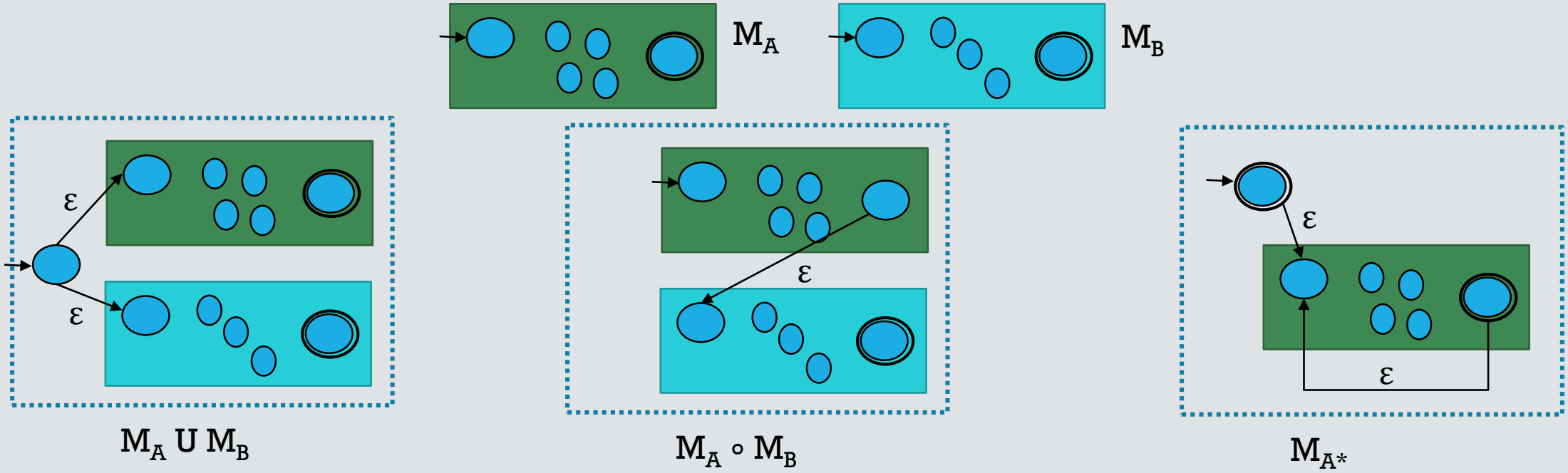


  - Example 2: M accepts strings over {a, b} if they contain either the substring "aa" or "bb"



Closure under regular operations

- Recall that "The class of regular languages is closed under the union (concatenation and star) operation"
  - Easier to proof using NFA
  - Let A and B be two regular languages

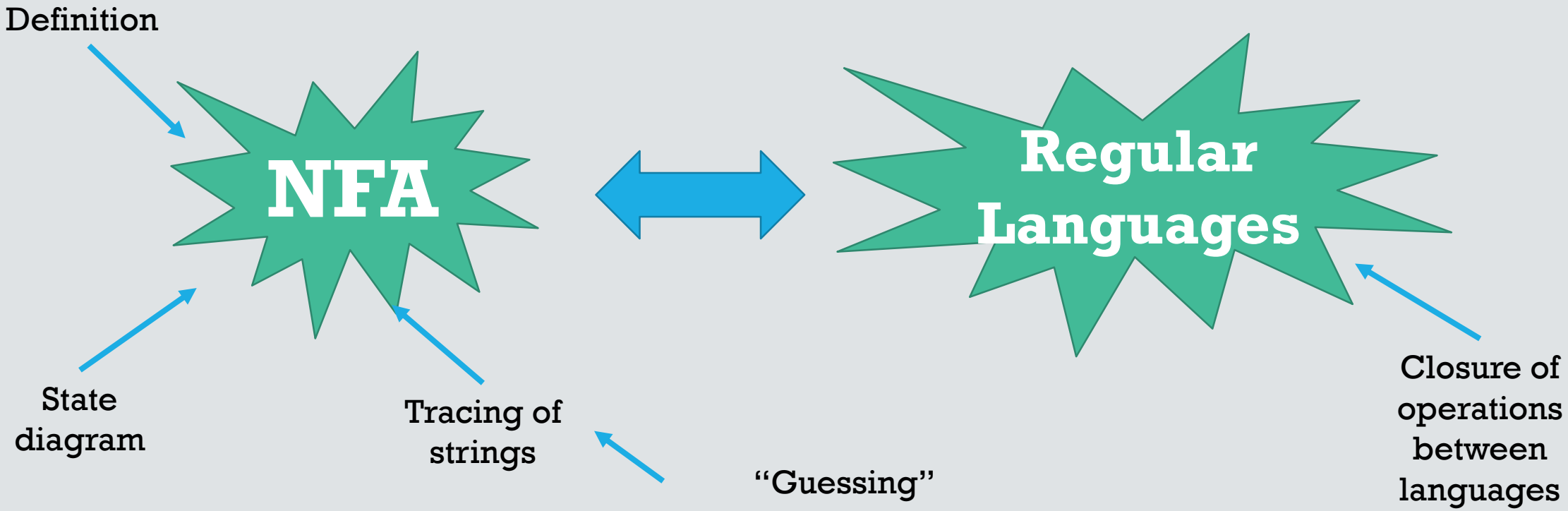


$M_A \cup M_B$ $M_A \circ M_B$ M_{A^*}

Closure Summary

Combing properties	Language operations	Operations on FAs
w starts with 0 **and** ends with 0	Intersection of L1 and L2 $L_1 \cap L_2$	"Cross product" of M1 and M2 **Note:** M1 and M2 must be DFAs
w starts with 0 **or** ends with 0	Union of L1 and L2 $L_1 \cup L_2$	Union of M1 and M2: 1. Create a new start state 2. Add ε transitions from the new start state to the old start states of M1 and M2
w has substring 11 **followed by** substring 01	Concatenation of L1 and L2 $L_1 \circ L_2$	Concatenation of M1 and M2: 1. Create an ε transition from the final states of M1 to the start state of M2. 2. Make the final states of M1 as non-final. 3 The start state of M1 is the new start state of the combined machine.
w has substring 101 **appeared any number of times**	Start operation of L1 L_1^*	Star operation on M1 1. Create an ε transition from the final states to the M1's start state 2. Create a new start state that is also a final state 3. Create an ε transition from the new start states to the old start state of M1

So Far…


Definition
NFA
Regular Languages
State diagram
Tracing of strings
"Guessing"
Closure of operations between languages

Equivalence of NFAs and DFAs

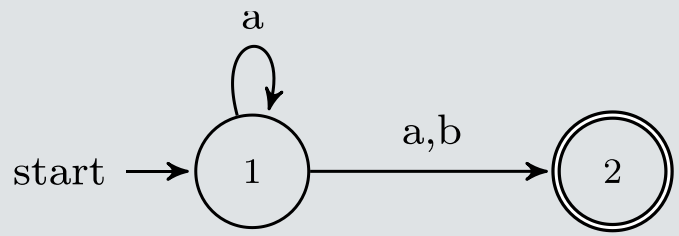
- Two machines are equivalent if they recognize the same language
- NFAs and DFAs recognize the same class of languages
- **Theorem 1.39**
  - Every NFA has en equivalent deterministic FA
  - Basically, for every NFA N, there exists a DFA M (that simulates N), such that $L(N)=L(M)$
  - Proof idea:
    - We'll assume that NFA N exists and we'll construct a DFA that simulates M that simulates N
    - In other words, we'll show that following certain steps you can convert an NFA into a DFA

Equivalence of NFAs and DFAs - Proof

- Let $N = (Q, \Sigma, \delta, q_o, F)$ be the NFA recognizing some language A. (Assume N does not have ε – transitions)
- Define $M = (Q', \Sigma, \delta', q_o', F')$, where
  1. $Q' = P(Q)$
     - Remember: since M is simulating N, at some point, at a step in the computation, M may point to several current states in N
  2. For $R \in Q'$ and for (each) a $\in \Sigma$
     - $\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a)$ for some $r \in R\}$
     - Another way to write this: $\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$
     - Remember: If R is a state of M, then it is a set of states in N. When M in state R processes an input symbol a, M goes to a set of states to which N will go in any of the branches of its computation.
  3. $q_o' = \{q_o\}$
  4. $F' = \{R \in Q' \mid R$ contains an accept state of $N\}$

NFA to DFA – A Simple Example

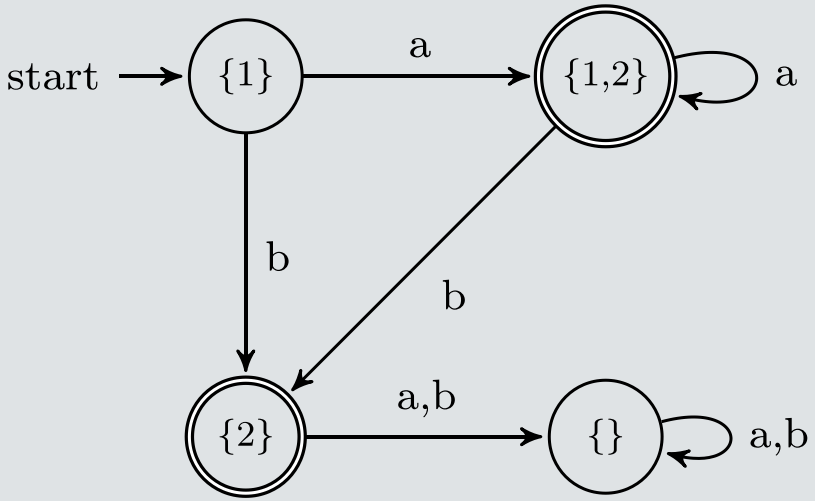
NFA N



State	a	b
1	{1,2}	{2}
2	{}	{}

$$\mathcal{P}(Q) = \{\{\}, \{1\}, \{2\}, \{1,2\}\}$$

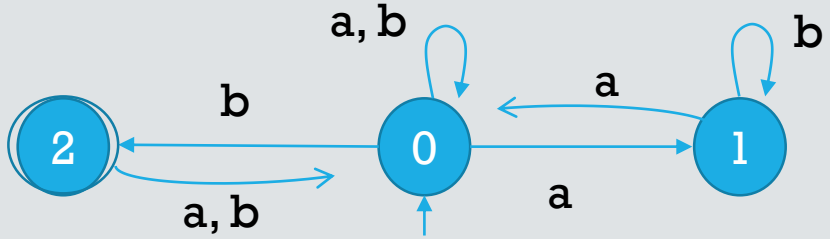
DFA M



State	a	b
{1}	{1,2}	{2}
{2}	{}	{}
{1,2}	{1,2}	{2}
{}	{}	{}

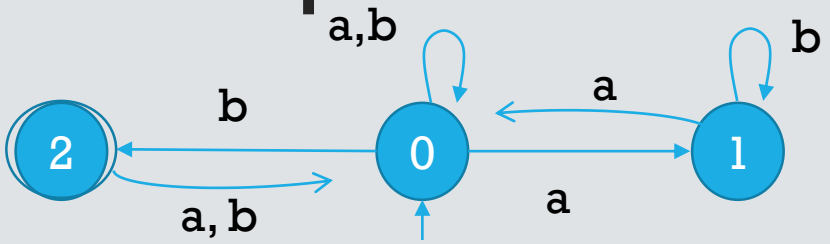
Example 1 - NFA to DFA

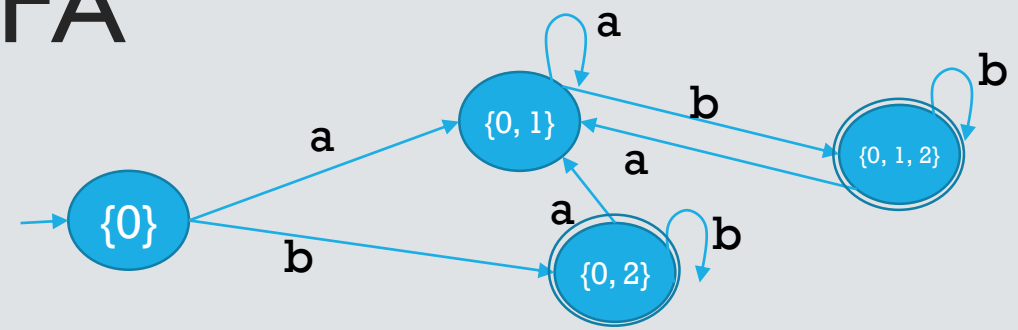
- Consider the following NFA, which accepts the string "ab"



  - To simulate this NFA, the DFA must remember the current state of each copy of this NFA after reading an input symbol. Because, in general, NFA may end up in any combination of its state, the set of DFA states is the power set of NFA states. For the above NFA its DFA equivalent contains the following states ∅,{0},{1},{2},{01},{0,2},{1,2},{0,1,2}
  - The start state is {0} since its where the NFA begins it simulation
  - The final states of DFA are those that contain NFA's state 2 (the final state of the NFA), i.e., {2},{0,2},{1,2} and {0,1,2}.

Example 1 - NFA to DFA

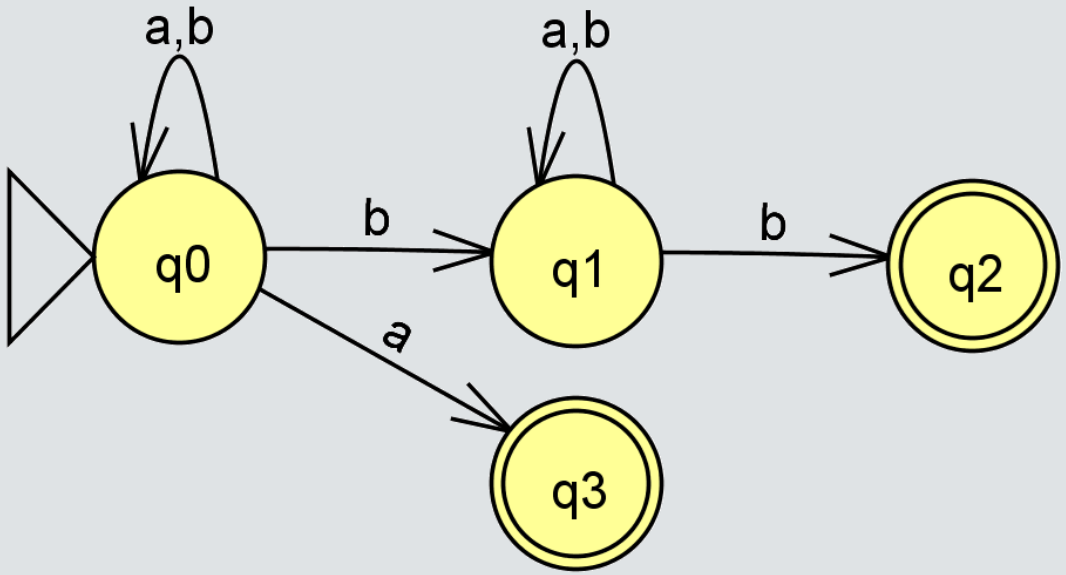




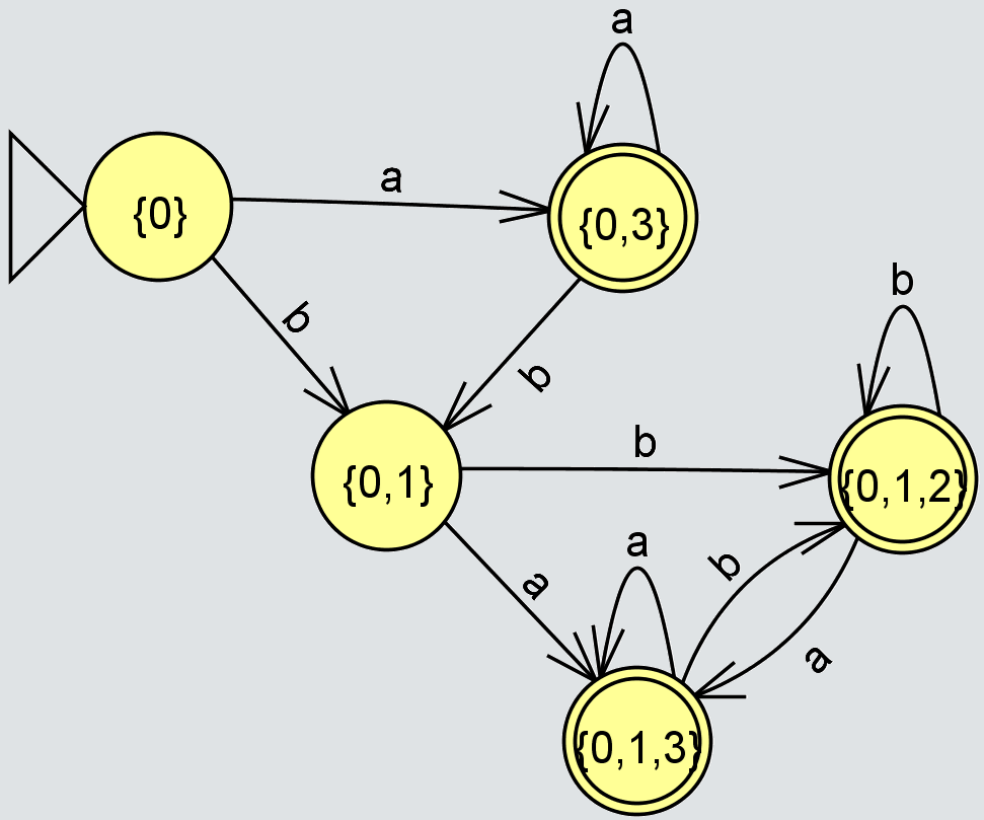
- Using the NFA above we can see that:
  - $\delta'(\{0\},a) = \{0,1\}$ and $\delta'(\{0\},b) = \{0,2\}$, i.e., on the input symbol a from the state 0 the NFA can go to states {0,1}, and on input symbol b from the same state the NFA can go to states {0,2}
  - $\delta'(\{0,1\},a) = \delta(\{0\},a) \cup \delta(\{1\},a) = \{0,1\} \cup \{0\} = \{0,1\}$ and $\delta'(\{0,1\},b) = \delta(\{0\},b) \cup \delta(\{1\},b) = \{0,2\} \cup \{1\} = \{0,1,2\}$
  - $\delta'(\{0,2\},a) = \delta(\{0\},a) \cup \delta(\{2\},a) = \{0,1\} \cup \{0\} = \{0,1\}$ and $\delta(\{0,2\},b) = \delta(\{0\},b) \cup \delta(\{2\},b) = \{0,2\} \cup \{0\} = \{0,2\}$
  - $\delta'(\{0,1,2\},a) = \delta(\{0\},a) \cup \delta(\{1\},a) \cup \delta(\{2\},a) = \{0,1\} \cup \{0\} \cup \{0\} = \{0,1\}$ and $\delta'(\{0,1,2\},b) = \delta(\{0\},b) \cup \delta(\{1\},b) \cup \delta(\{2\},b) = \{0,2\} \cup \{1\} \cup \{0\} = \{0,1,2\}$
- Using the "Table" strategy
  - * indicates final state

States	a	b
0	0 ,1	0, 2
1	0	1
2 *	0	0
{0}	{ 0,1}	{0,2}
{ 0,1}	{ 0,1}	{0,1,2}
{0,2} *	{ 0,1}	{0,2}
{0,1,2} *	{ 0,1}	{0,1,2}

Example 2 - NFA to DFA

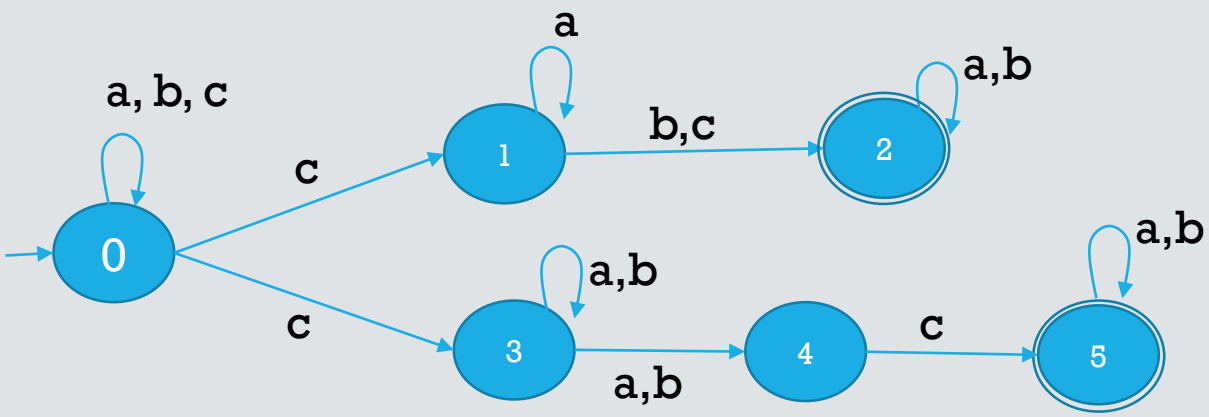


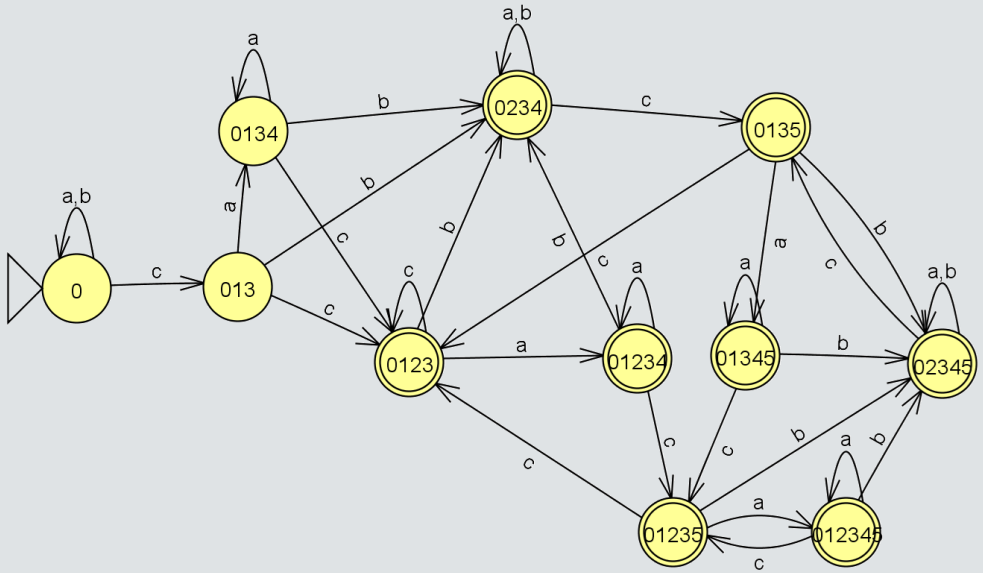
Nondeterministic FA



Deterministic FA

Example 3 - NFA to DFA

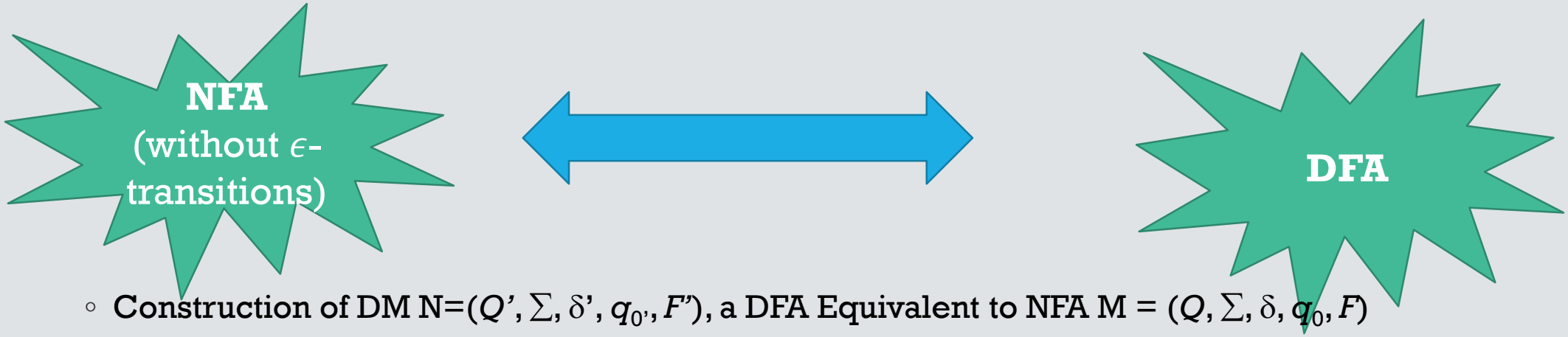




Nondeterministic FA

Deterministic FA

So Far…



- Construction of DM N=(Q', $\sum$, δ', $q_{0'}$, F'), a DFA Equivalent to NFA M = (Q, $\sum$, δ, q_0, F)

1. Initialize Q' to $\{q_0\}$
2. Repeat

   IF there is a node $X \in Q'$ and a symbol $a \in \sum$ with no arc leaving X labeled a, THEN

   a) Let $Y = \cup_{q_i \in X} \delta(q_i, a)$
   b) IF $Y \notin Q'$, THEN set $Q' = Q' \cup \{ Y \}$
   c) Add an arc from X to Y labeled a

   ELSE *done* := true

   UNTIL *done*
3. The set of accepting states of DM is $F' = \{ X \in Q' \mid X \text{ contains } q_i \in F \}$

Equivalence of NFAs and DFAs

- Proof
  - Let $N = (Q, \Sigma, \delta, q_o, F)$ be the NFA recognizing some language A. **(Assume N has ε – transitions)**
  - Define $M = (Q', \Sigma, \delta', q_o', F')$, where
    1. $Q' = P(Q)$
    2. $F' = \{R \in Q' \mid R$ contains an accept state of $N\}$
    3. $q_o' = \boldsymbol{E(\{q_o\})}$, where ***E(R)*** is the collection of states that can be reached from R by ε – *transitions*, including the members of R themselves
    4. For $R \in Q'$ and for (each) a $\in \Sigma$, $\delta'(R, a) = \{q \in Q \mid q \in E(\delta(r, a))$ for some $r \in R\}$

Equivalence of NFAs and DFAs

- Construction of DM N=(Q', $\sum$, δ', $q_{0'}$, F'), a DFA Equivalent to NFA M = (Q, $\sum$, δ, q_0, F)

1. Initialize Q' to {E(q_0)}
2. Repeat

   IF there is a node $X \in Q'$ and a symbol $a \in \sum$ with no arc leaving X labeled a, THEN

   a) Let $Y = \cup_{q_i \in X} E(\delta(q_i, a))$
   b) IF $Y \notin Q'$, THEN set $Q' = Q' \cup \{ Y \}$
   c) Add an arc from X to Y labeled a

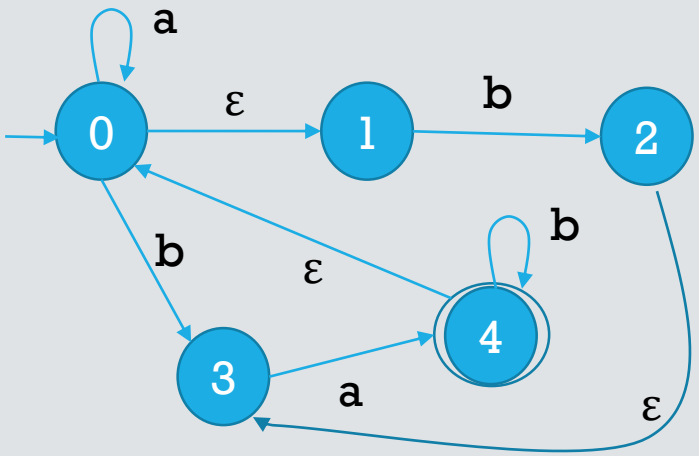
   ELSE *done* := true

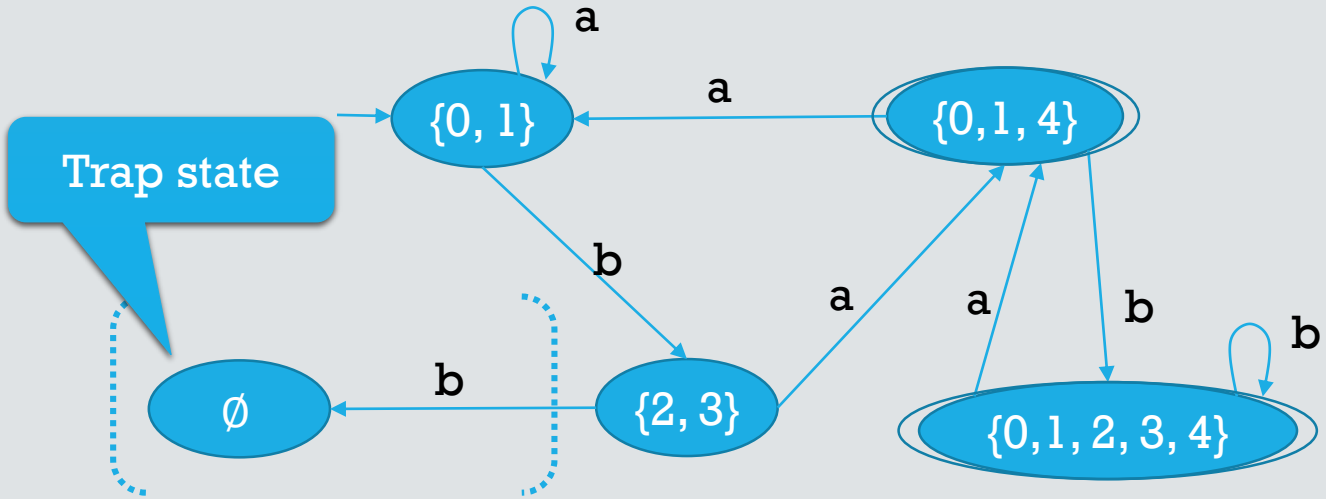
   UNTIL *done*
3. The set of accepting states of DM is $F' = \{ X \in Q' \mid X \text{ contains } q_i \in F \}$

Remember!

- E(q)={states that can be reached from q by travelling along 0 or more ε – transitions}

Example 4 - NFA to DFA

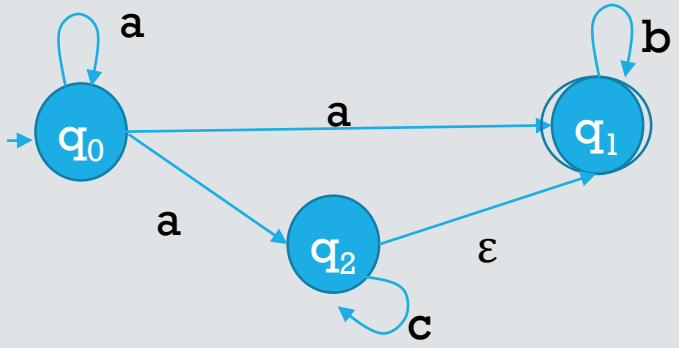


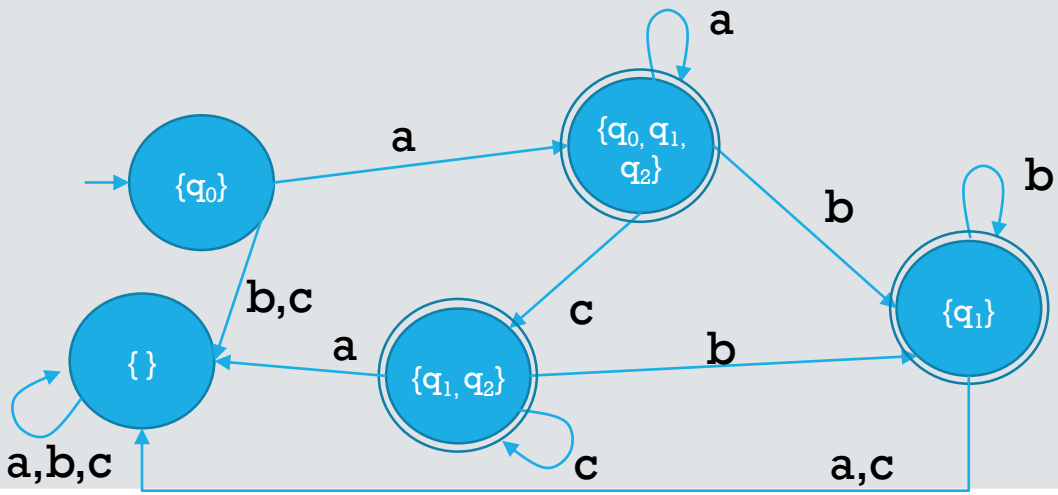


- E(0) ={0} ∪ {1} = {0, 1}
- $\delta'(\{0,1\}, a) = E(\delta(0, a) \cup \delta(1, a))$=$E(\{0\} \cup \emptyset)$= $E(\{0\})$={0,1}
- $\delta'(\{0,1\}, b) = E(\delta(0, b) \cup \delta(1, b))$=$E(\{3\} \cup \{2\})$= $E(\{2, 3\})$={2, 3}
- $\delta'(\{2,3\}, a) = E(\delta(2, a) \cup \delta(3, a))$=$E(\emptyset \cup \{4\})$= $E(\{4\})$={0, 1, 4}
- $\delta'(\{2,3\}, b) = E(\delta(2, b) \cup \delta(3, b))$=$E(\emptyset \cup \emptyset)$= $E(\emptyset)$=$\emptyset$
- $\delta'(\{0, 1, 4\}, a) = E(\delta(0, a) \cup \delta(1, a) \cup \delta(4, a))$=$E(\{0\} \cup \{1\} \cup \{4\})$= $E(\{0\} \cup \emptyset \cup \emptyset)$= $E(\{0\})$={0,1}
- $\delta'(\{0, 1, 4\}, b) = E(\delta(0, b) \cup \delta(1, b) \cup \delta(4, b))$=$E(\{3\} \cup \{2\} \cup \{4\})$= $E(\{2, 3,4\})$= {0,1,2,3,4}

Example 5 - NFA (with ε – transitions) to DFA

- Consider the following NFA





Transitions from NFA

	a	b	c
q_0	$\{q_0,q_1,q_2\}$	{ }	{ }
q_1	{ }	$\{q_1\}$	{ }
q_2	{ }	$\{q_1\}$	$\{q_1,q_2\}$

Transitions for DFA

$\{q_0\}$ (s)		$\{q_0,q_1,q_2\}$	{ }	{ }
$\{q_0,q_1,q_2\}$	f	$\{q_0,q_1,q_2\}$	$\{q_1\}$	$\{q_1,q_2\}$
$\{q_1\}$	f	{ }	$\{q_1\}$	{ }
$\{q_1,q_2\}$	f	{ }	$\{q_1\}$	$\{q_1,q_2\}$

Equivalence of NFAs and DFAs

- Theorem 1.39 state the every NFA can be converted into a equivalent DFA
- Corollary
  - A language is **regular** if an only if some nondeterministic finite automaton recognizes it

Regular Expressions

- A **regular expression (RE)** is a string of symbols that describes a regular language
  - RE provide a powerful method for describing patterns
  - Simpler alternative to describe the language recognized by a model (e.g., FA)
  - Definition
    - R is a regular expression if R is
      1. a for some a in the alphabet Σ
      2. ε
      3. $\emptyset$
      4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions
      5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions
      6. (R_1^*), where R_1 is a regular expression
  - Precedence of operations
    - Star
    - Concatenation
    - Union

Regular Expressions

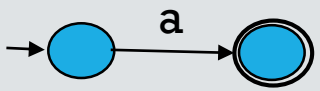
- Examples
  - $0^*10^* = \{ w \text{ over } \{0, 1\} \mid w \text{ contains a single } 1\}$
  - $\Sigma^*aba\Sigma^* = \{w \text{ over } \Sigma \mid w \text{ contains aba as its substring}\}$
  - $(0 \cup \varepsilon)1^* = 01^* \cup 1^*$
  - $1^*\emptyset = \emptyset$
  - $a^*b^* =$ any string without substring ba
  - $((a \cup c)^*b(a \cup c)^*b(a \cup c)^*b)^+ =$ strings where the number of b's is a multiple of 3

Regular Expressions

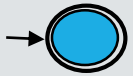
- Observations
  - RE ε represents the language containing a single string, i.e., the empty string
  - RE $\emptyset$ represents the language that doesn't contain any strings
  - $R^+ = RR^*$ (one or more concatenations of strings from R)
    - Since R^* refers to zero or more concatenations of string from R, then $R^+ \cup \{\varepsilon\} = R^*$
  - The language represented by R is denoted L(R)
  - $R \cup \emptyset = R$
  - $R \circ \varepsilon = R$
  - $R \cup \varepsilon = R$? Not always. R=0 then L(R)={0} but L(R U ε)={0, ε}
  - $R \circ \emptyset = R$? Not always. R=0 then L(R)={0} but $L(R \circ \emptyset) = \emptyset$
  - $\emptyset^* = \varepsilon$

Equivalence with FAs

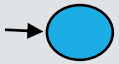
- Theorem
  - A language is regular if and only if some regular expression describes it
- Lemma
  - If a language is described by a regular expression, then it is regular
- Proof idea:
  - We have a RE R describing a language A and we show how to convert R into a NFA that recognizes A. By corollary 1.40, if NFA recognizes A, then A is regular
- Proof: by construction, on each of the six cases in the formal definition of RE



R = a



R = ε



R = ∅

$R_1 \cup R_2$

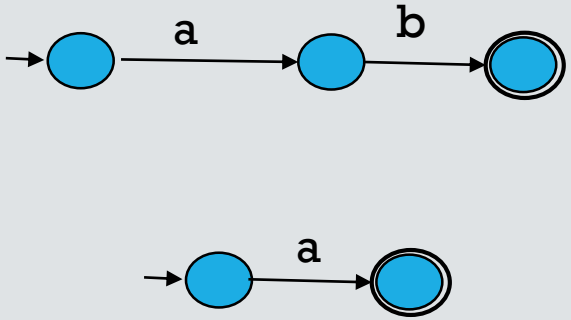
$R_1 \circ R_2$

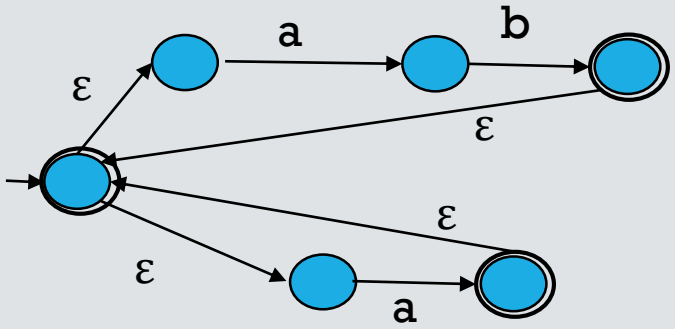
R_1^*

Illustrated on NFAs on slide 56

Equivalence with FAs

- Example
  - Show that (ab U a)* represents a regular language





  - Show that (a U b)*aba represents a regular language

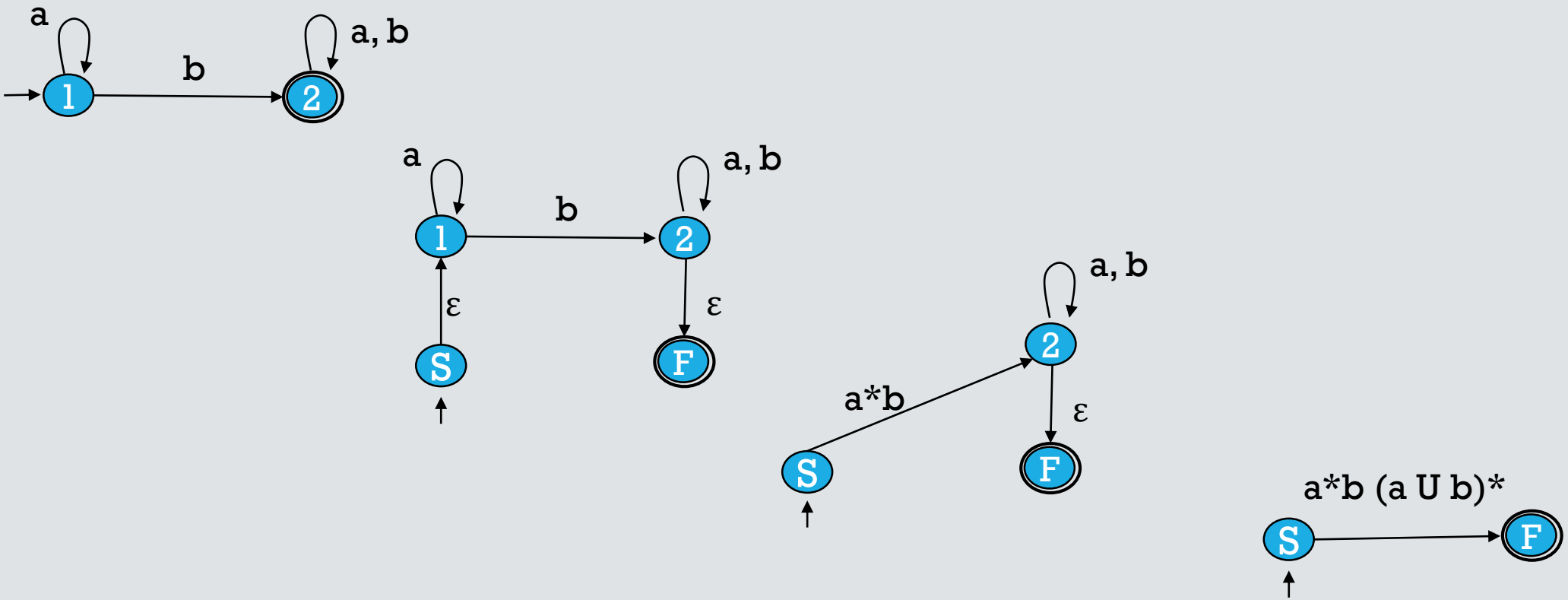
NFA to RE

- Lemma 1.60
  - If a language is regular, then it is described by a regular expression
  - Proof idea: Describe the procedure for converting FAs into a equivalent regular expressions
- Generalized Nondeterministic Finite Automata (GNFA)
  - Very similar to an NFA, except that a label of a GNFA can be a regular expression
  - Can read blocks of symbols from the input where symbols are strings of a regular expression
- Example
  - GNFA accepts all strings over {a, b} with substring aba or aa

GNFA

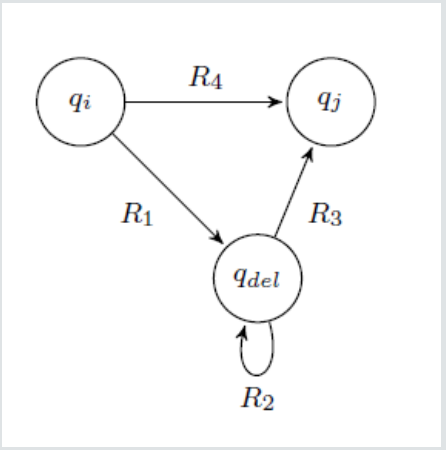
- Basic idea
  1. Convert FA into GNFA which satisfies the following conditions
     - The start state has only outgoing transitions, but does not have any incoming/loopback transitions
     - There is only a single final state, which has only incoming transitions, but does not have any outgoing/loopback transitions
  2. Eliminate the states of the GNFA one by one (except start and final states), and update the transitions accordingly, so that the new GNFA recognizes the same language
     - From a state to any other state there should be at most one transition
  3. When only the start state and the final state are left, the transition from the start state to the final state is an equivalent regular expression

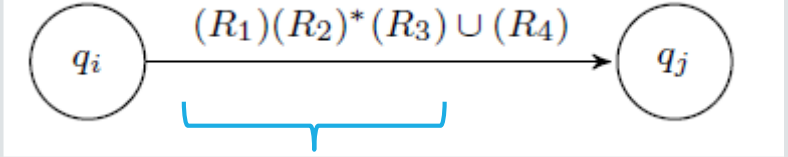
GNFA – Example 1



GNFA – Example 2

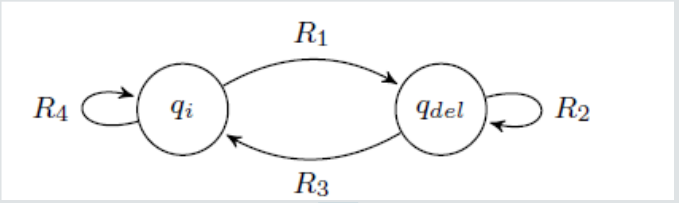
- General Rule for State Elimination
  - Consider three states q_i, q_j and q_{del} where the latter is the state we want to eliminate
  - Let the transitions between those three states be as follows
    - R1 = (q_i, q_{del})
    - R2 = (q_{del}, q_{del})
    - R3 = (q_{del}, q_j)
    - R4 = (q_i, q_j), where R is some regular expression describing that transition

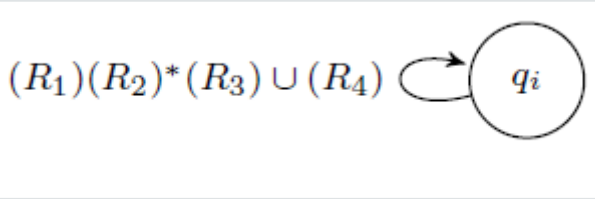




Original RE from q_i to q_j

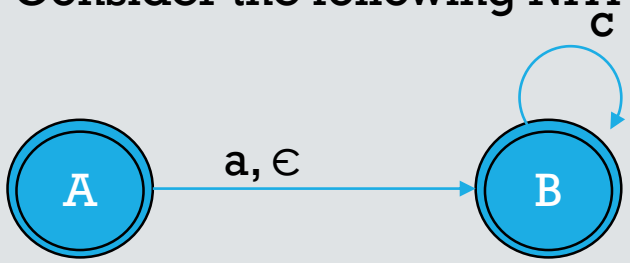
RE resulting from removing q_{del}





GNFA – Example 3

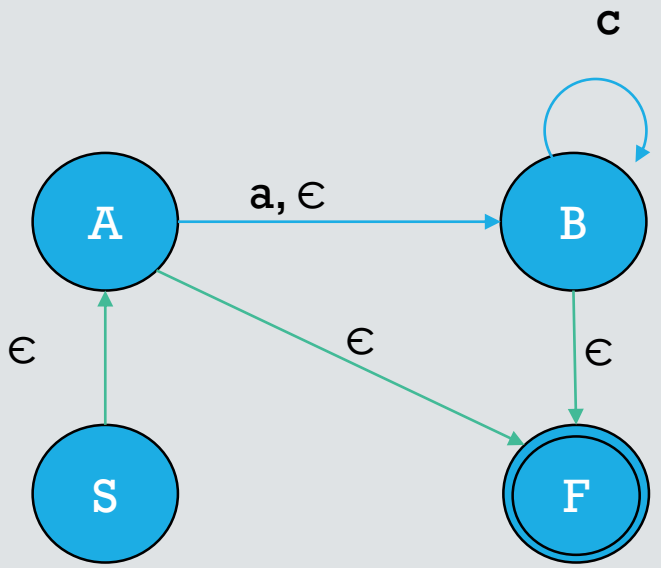
- Consider the following NFA *E*



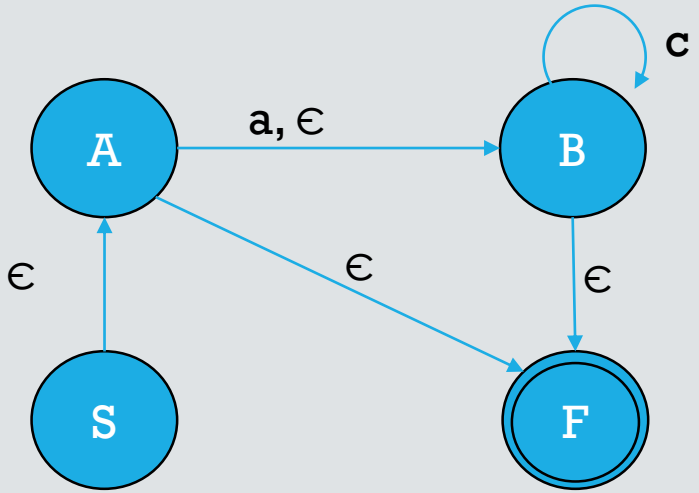
Start GNFA

New start state S

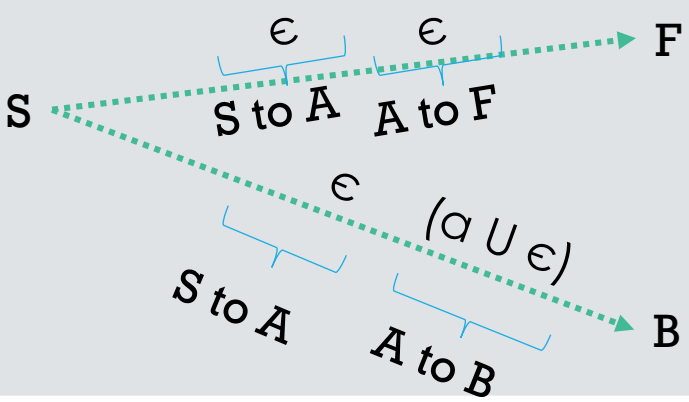
New, single, final state F

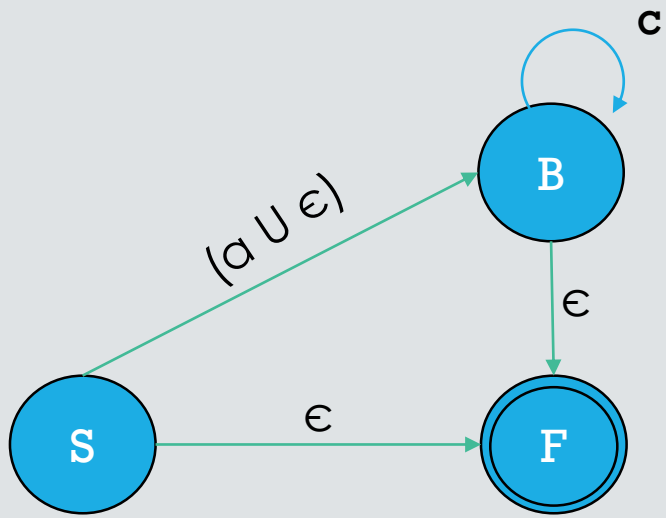


GNFA – Example 3

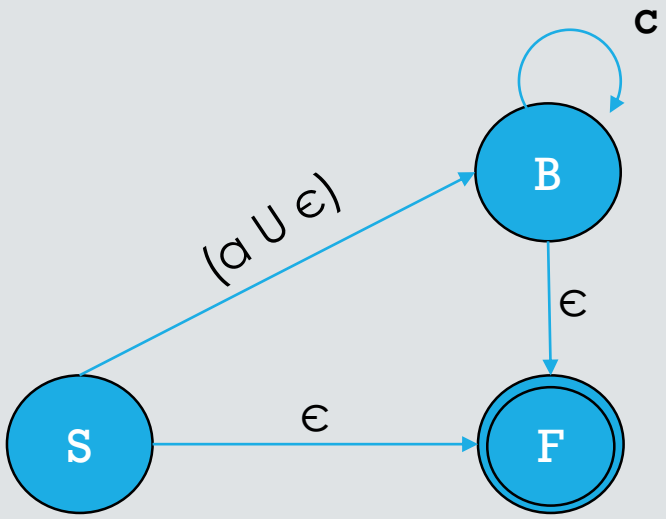


Remove A

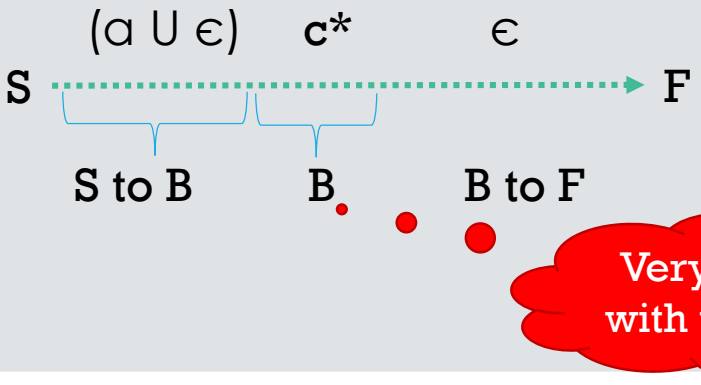


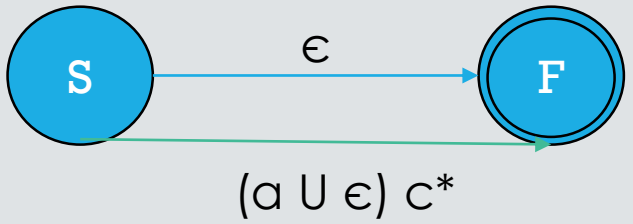


GNFA – Example 3



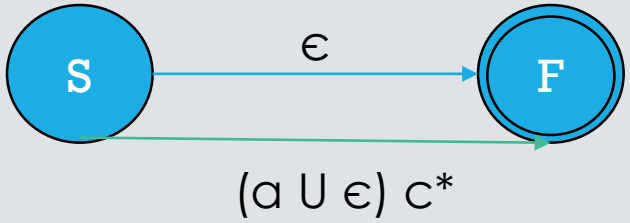
Remove B





Very careful with the loop!

GNFA – Example 3



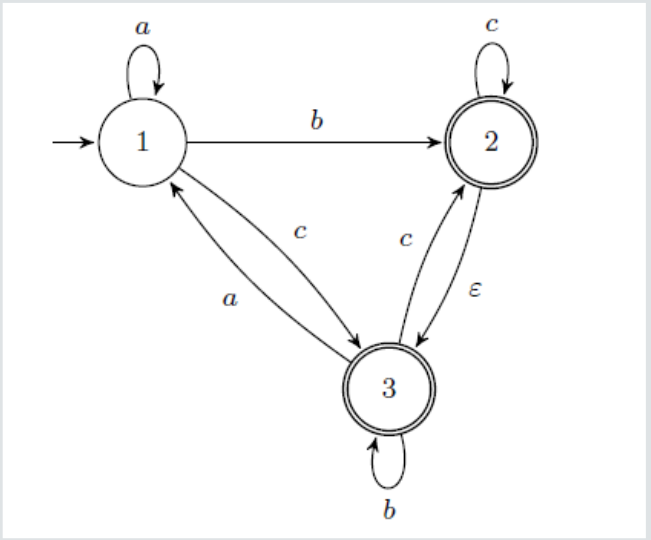
Regular Expression capturing the language accepted by for NFA E

(a U ϵ) c* U ϵ

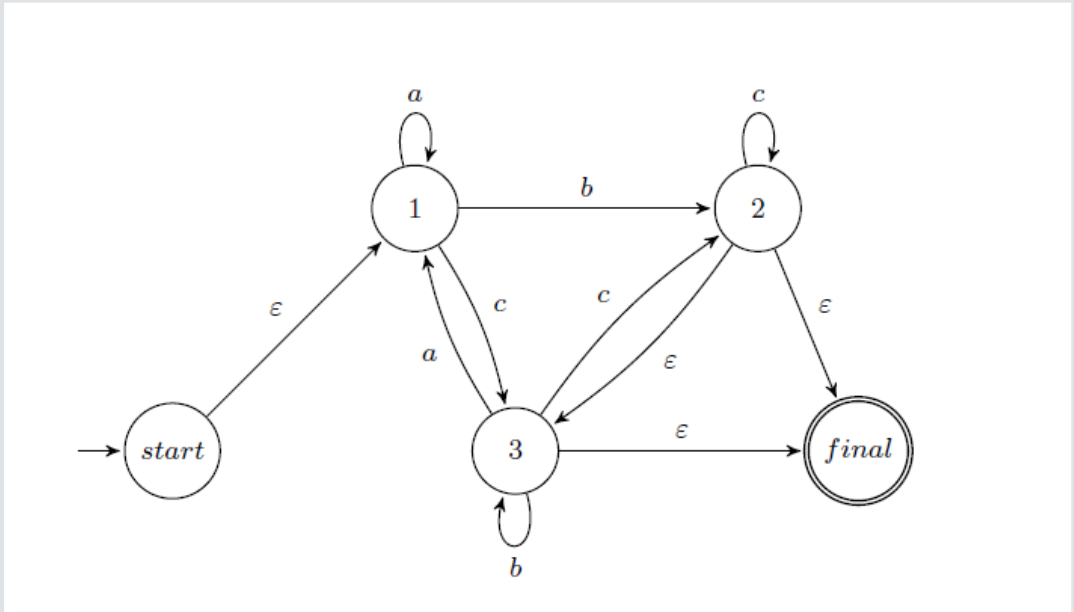
Very careful!
You need to account for
both paths from a to S

GNFA – Example 4

- What is the regular expressions of the language recognized by the following FA?

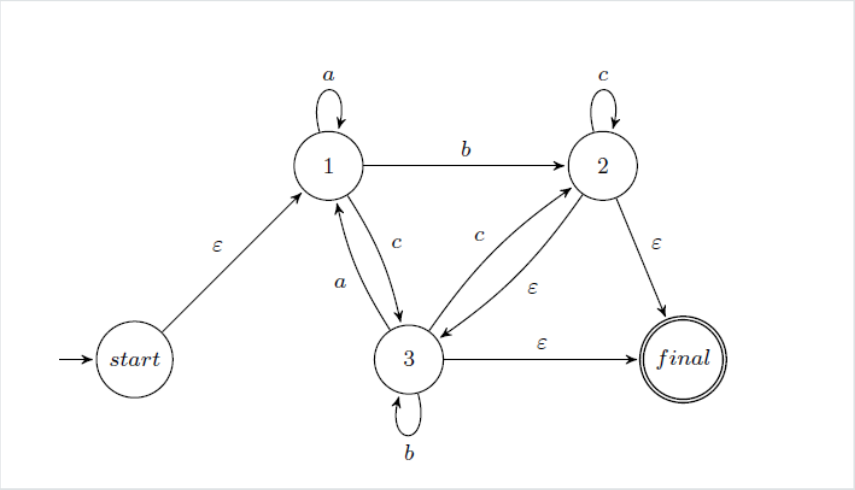


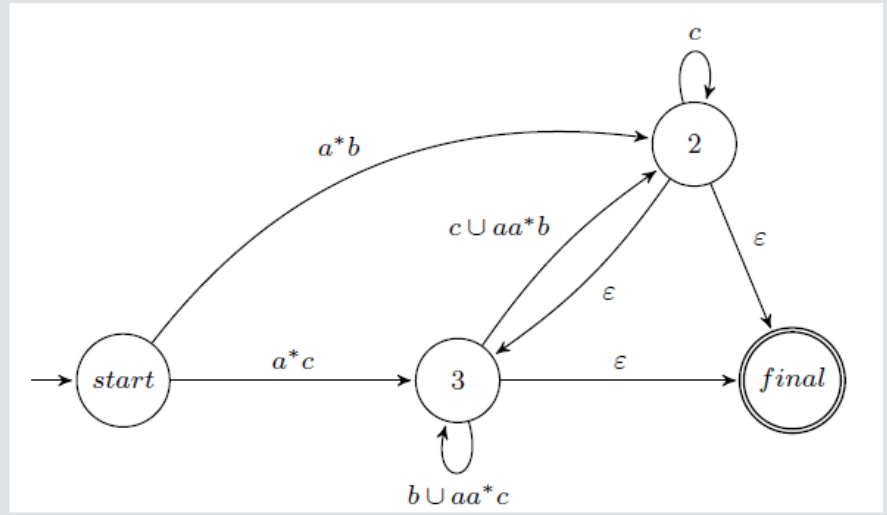
Add a new start/final state



GNFA (with a single start and single final state)

GNFA – Example 4



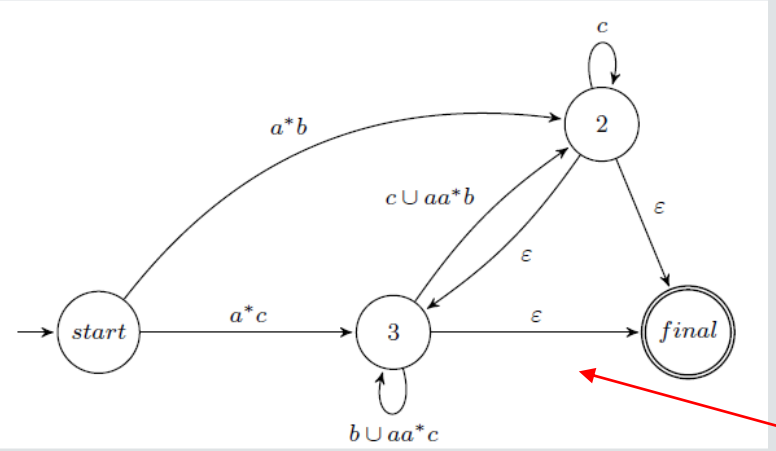


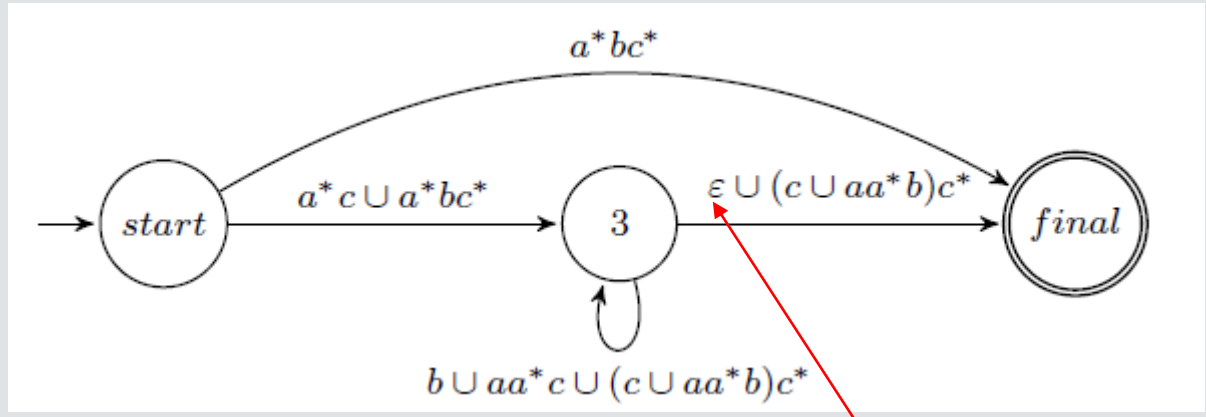
For removing state 1

(2 incoming arrows, 2 outgoing arrows, excluding any transition to itself => 2x2=4 possible paths)

- path 1: start→1→2 : $\epsilon a^*b = a^*b$ (new transition from start→2)
- path 2: start→1→3 : $\epsilon a^*c = a^*c$ (new transition from start→3)
- path 3: 3 →1→2 : aa^*b (added using the ∪ operator on the existing transition from 3->2)
- path 4: 3 →1→3 : aa^*c (added using the ∪ operator on the existing transition from 3->3)

GNFA – Example 4

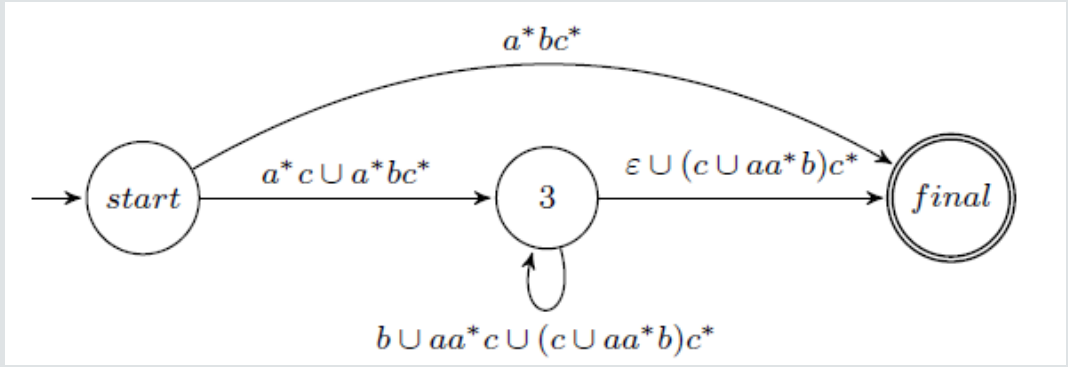




For removing state 2
(2 incoming arrows, 2 outgoing arrows, excluding any transition to itself => 2x2=4 possible paths)

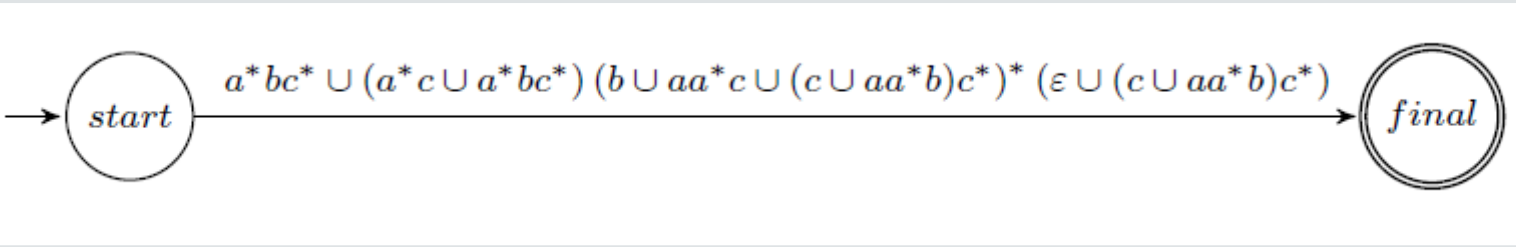
- path 1: start→2→3 : $a^*bc^*\epsilon = a^*bc^*$ (added to existing transition start→3)
- path 2: start→2→final : $a^*bc^*\epsilon = a^*bc^*$ (new transition from start→final)
- path 3: 3 →2→3 : $(c \cup aa^*b)c^*\epsilon = (c \cup aa^*b)c^*$ (added to existing transition 3→3)
- path 4: 3 →2→final : $(c \cup aa^*b)c^*\epsilon = (c \cup aa^*b)c^*$ (added to existing transition 3→final; the original ϵ that is already on the 3→final transition will be kept)

GNFA – Example 4

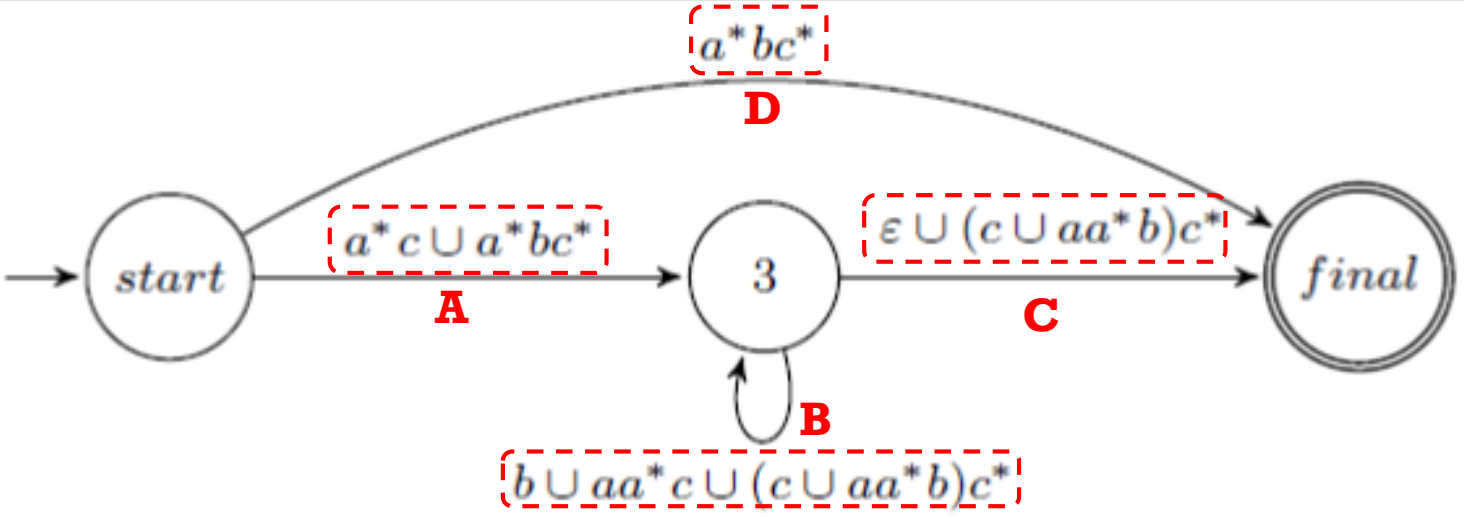


For removing 3

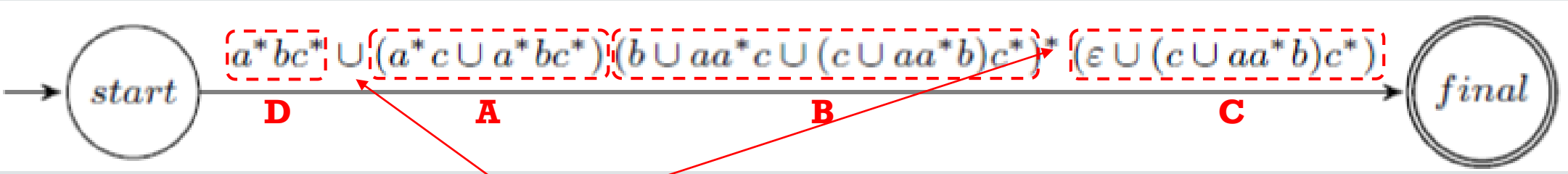
- path 1: start → final
- path 2: start → 3 → final



GNFA – Example 3



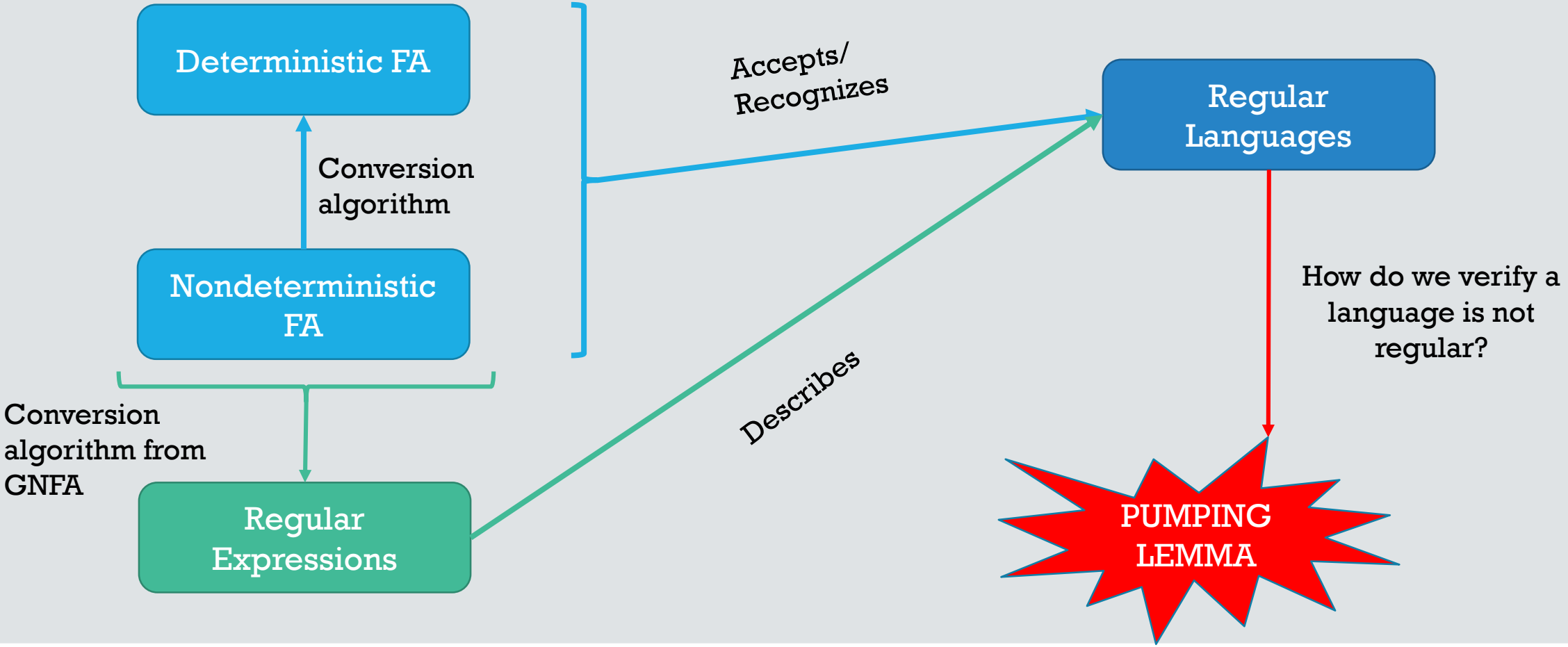
Remove 3



For removing state 3
(1 incoming arrow, 1 outgoing arrow, excluding any transition to itself => 1x1=1 possible path)

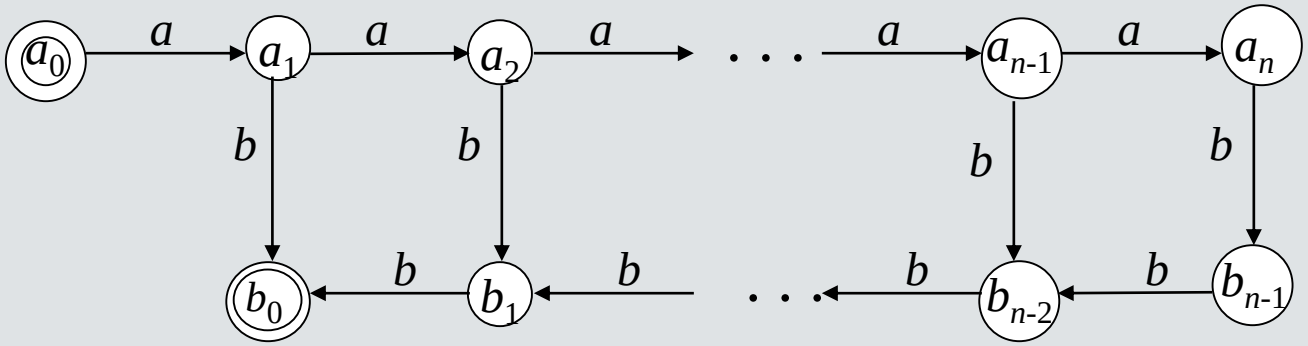
- path 1: start→3→final : AB^*C (added using the ∪ operator on the existing transition start→final, which already contains D)

So Far…


Deterministic FA
Conversion algorithm
Nondeterministic FA
Conversion algorithm from GNFA
Regular Expressions
Accepts/ Recognizes
Describes
Regular Languages
How do we verify a language is not regular?
PUMPING LEMMA

Non-Regular Languages

- Limitation for FA:
  - Can't deal with languages that require "remembering" an infinite number of possibilities, on a finite number of states
  - Example
    - The language $\{a^i b^i \mid 0 \le i \le n\}$ is regular, but the language $\{a^i b^i \mid i \ge 0\}$ is not
    - FA cannot be extended to accept $\{ a^i b^i \mid i \ge 0 \}$ since an infinite number of states are needed



Pumping Lemma for Regular Languages

- Pumping Lemma
  - Technique for proving nonregularity
    - The idea is to show that if a language doesn't have a known property of regular languages, then it is nonregular
      - The property states that string in the language can be "pumped" if they are of a certain length
      - Basically, if a section of a string can be repeated and the resulting string remains in the language, then we are dealing with a regular language