CS 302.1 - Automata Theory

Lecture 02

Shantanav Chakraborty

Center for Quantum Science and Technology (CQST)

Center for Security, Theory and Algorithms (CSTAR)

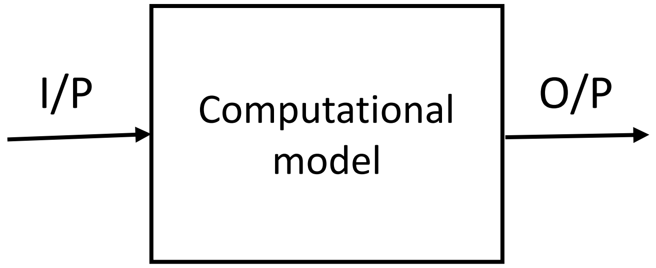
IIIT Hyderabad

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

A quick recap

- Can a given problem be computed by a particular computational model?

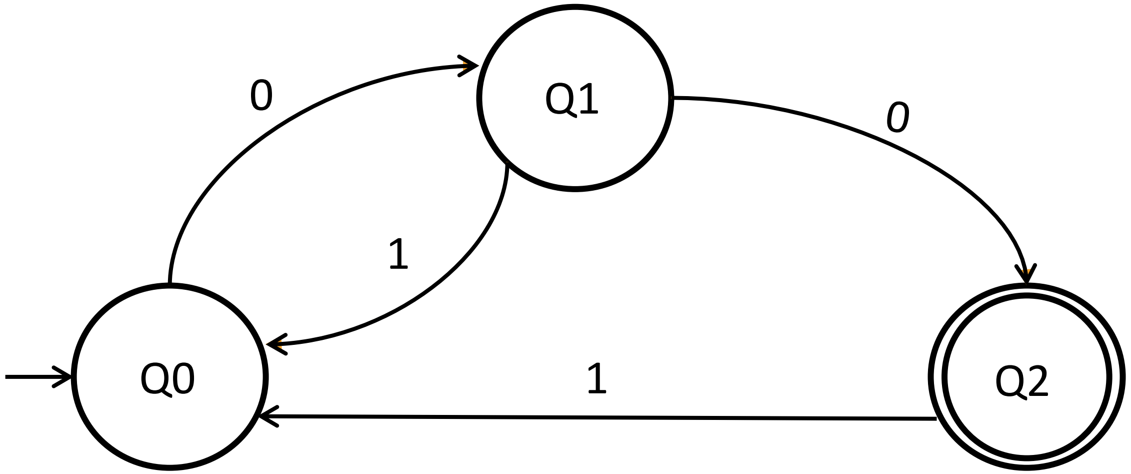


A computational model solves a problem P if,

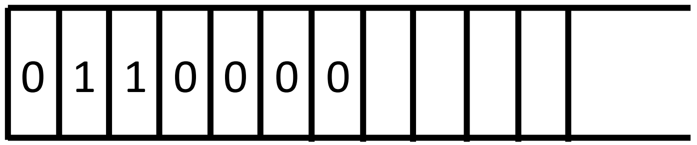
(i) For all inputs belonging to the YES instance of P, the device outputs **YES**

(ii) For all inputs belonging to the NO instance of P, the device outputs **NO**.

If (i) and (ii) hold, we say that the problem **P is computable** by this computational model.



Deterministic Finite Automata (DFA)

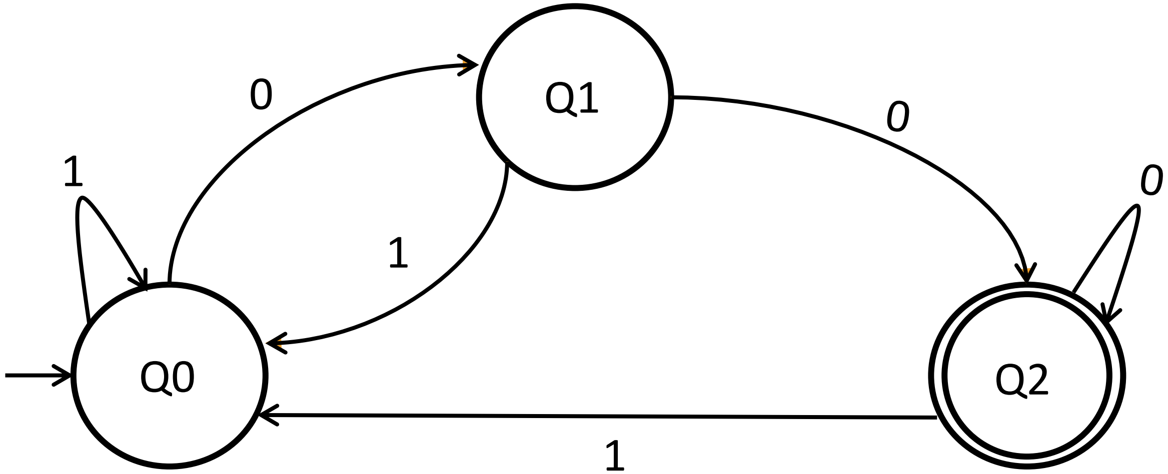


Run:

$$\boldsymbol{Q0} \xrightarrow{0} Q1 \xrightarrow{1} Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q1 \xrightarrow{0} Q2 \xrightarrow{0} Q2 \xrightarrow{0} \boldsymbol{Q2}$$

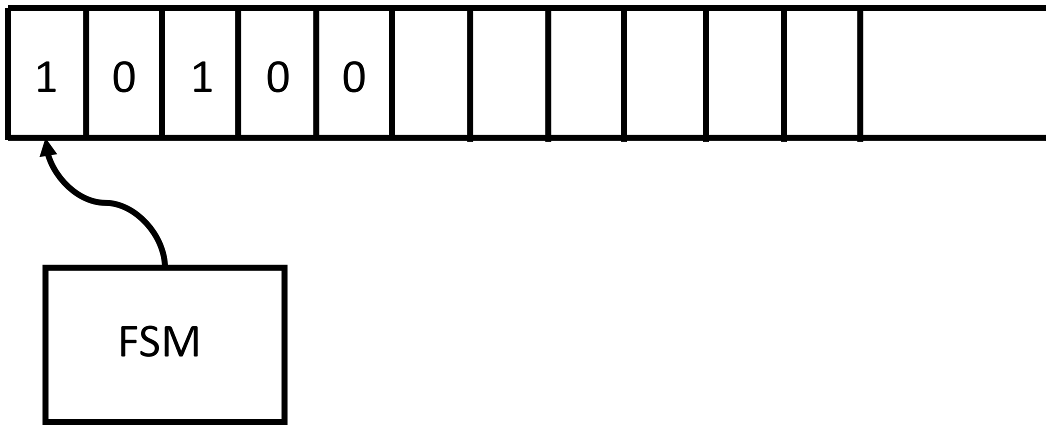
L(M) = {$\omega | \omega$ results in an accepting run}

Deterministic Finite Automata (DFA)



State transition diagram of the Finite State Machine

One-way infinite tape



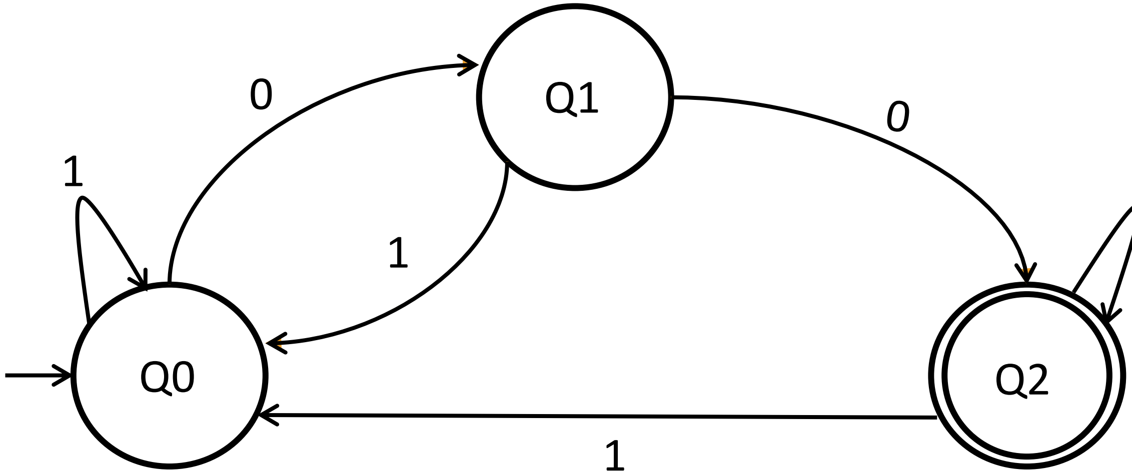
ACCEPT = {0111000, 10100, 0100, 00, 10000….}
REJECT = {11101, 0, 1, 11, 001,…….}

For any language L, we say **M recognizes L if**

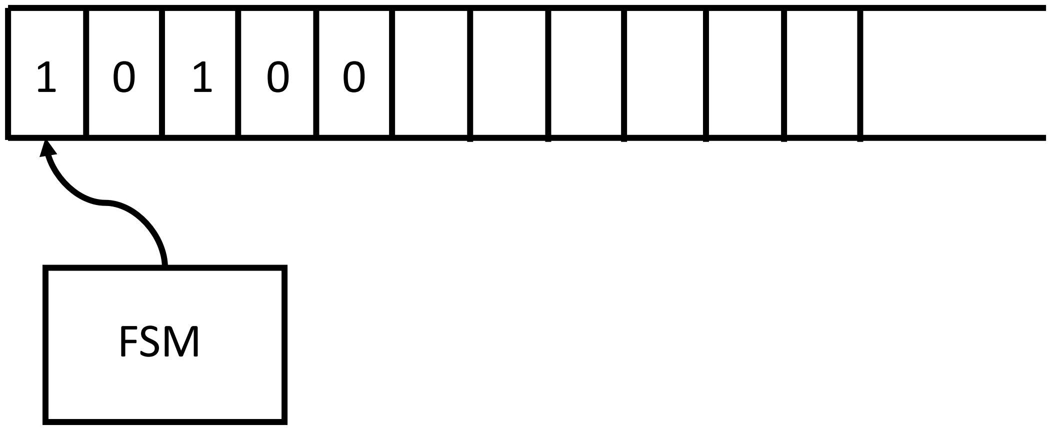
$$\forall \boldsymbol{\omega} \in \boldsymbol{L}, \boldsymbol{M}(\boldsymbol{\omega})$$ **accepts**

For the example above, $\boldsymbol{M}$ **recognizes L= {$\omega|\omega$ ends in "00"}**

Deterministic Finite Automata (DFA)



State transition diagram of the Finite State Machine



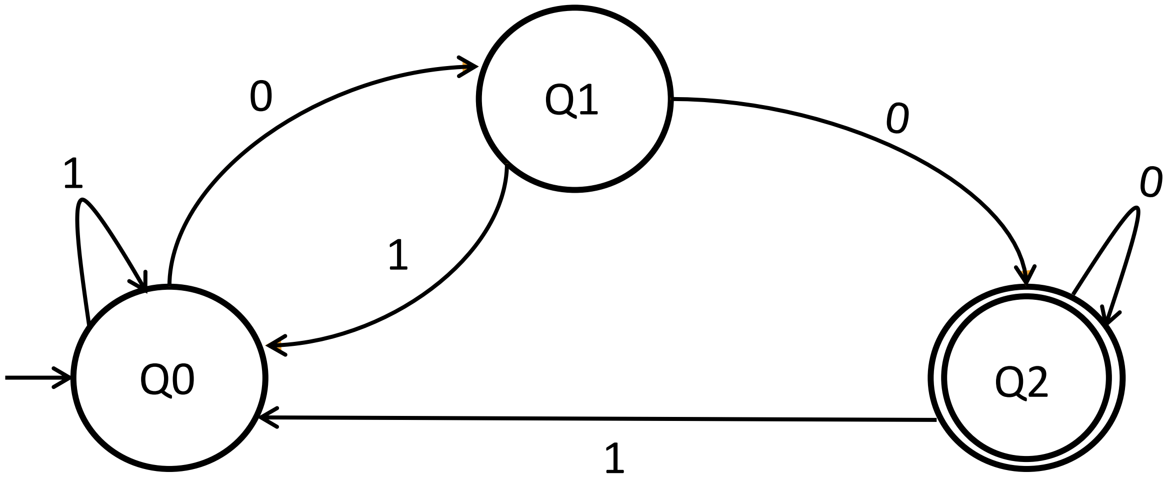
ACCEPT = {0111000, 10100, 0100, 00, 10000….}
REJECT = {11101, 0, 1, 11, 001,…….}

For any language L, we say **the problem M solves or M decides L if**

$$\forall \boldsymbol{\omega} \in \boldsymbol{L}, \boldsymbol{M}(\boldsymbol{\omega}) \textbf{ accepts}$$
$$\forall \boldsymbol{\omega} \notin \boldsymbol{L}, \boldsymbol{M}(\boldsymbol{\omega}) \textbf{ rejects}$$

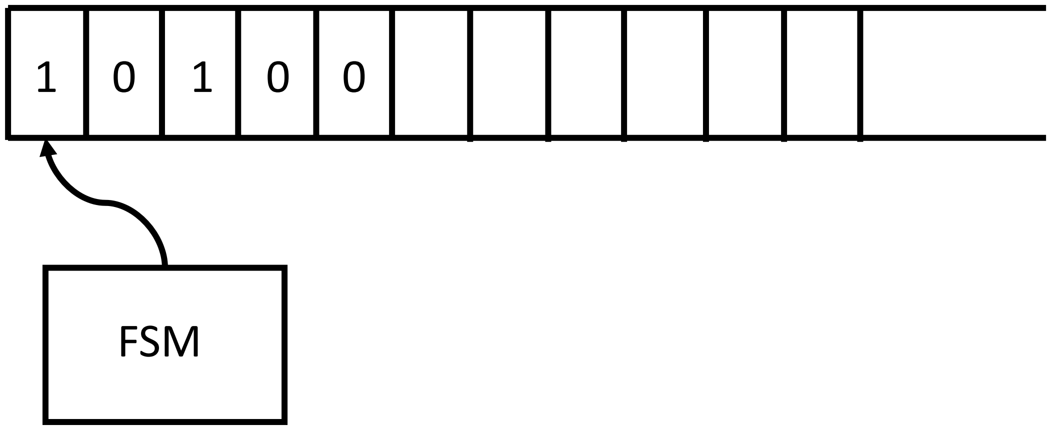
For the example above, $\boldsymbol{M}$ **decides L= {$\omega|\omega$ ends in "00"}**

Deterministic Finite Automata (DFA)



State transition diagram of the Finite State Machine

One-way infinite tape



For any language $\boldsymbol{L}$, we say **M recognizes L if**

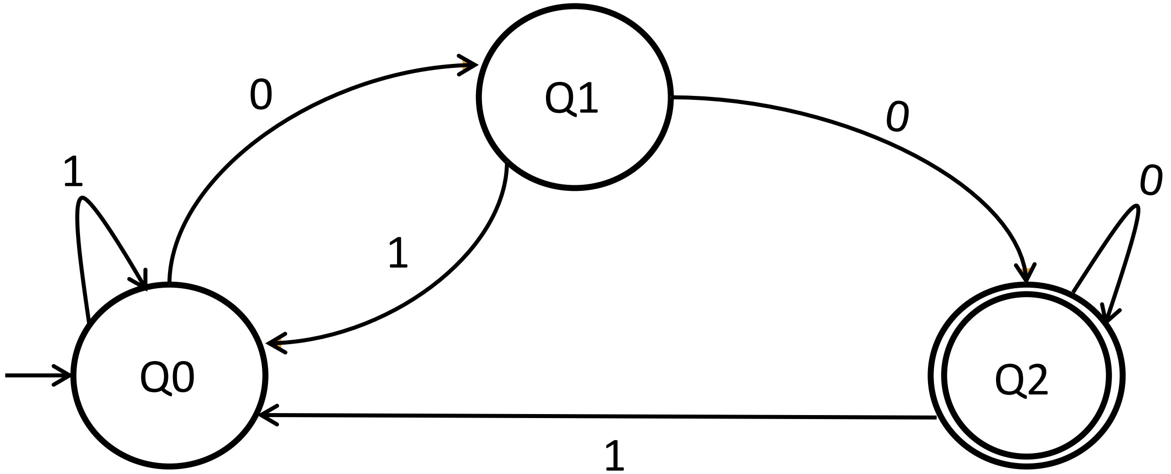
$$\forall \boldsymbol{\omega} \in \boldsymbol{L}, \boldsymbol{M}(\boldsymbol{\omega}) \text{ accepts}$$

For any language $\boldsymbol{L}$, we say **M decides L if**

$$\forall \boldsymbol{\omega} \in \boldsymbol{L}, \boldsymbol{M}(\boldsymbol{\omega}) \text{ accepts}$$
$$\forall \boldsymbol{\omega} \notin \boldsymbol{L}, \boldsymbol{M}(\boldsymbol{\omega}) \text{ rejects}$$

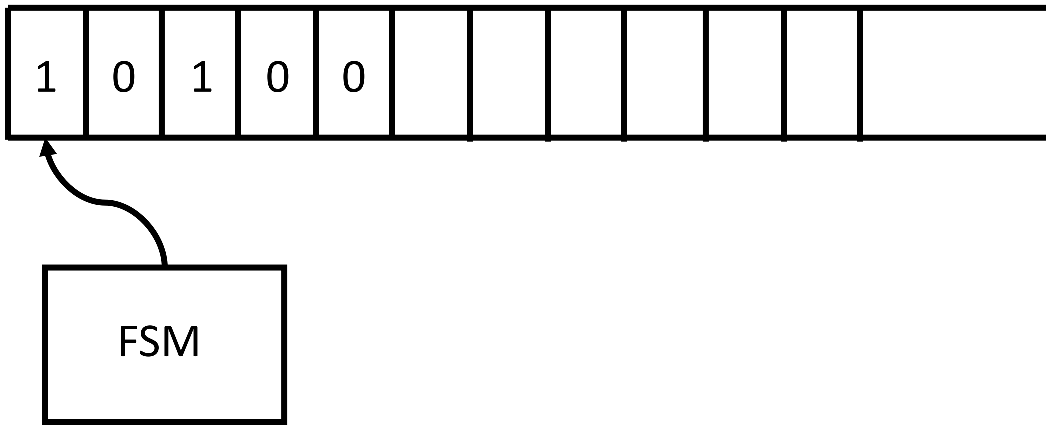
For a DFA, the notions of **deciding a language** and **recognizing a language** are equivalent, but this may not be true for other, more powerful computational models

Deterministic Finite Automata (DFA)



State transition diagram of the Finite State Machine

One-way infinite tape



Characteristics of DFA : (i) Single start state (ii) Unique transitions (iii) Zero or more final states

Formally, a finite automaton M is a 5-tuple $(\mathrm{Q}, \Sigma, \delta, q_0, F)$ where

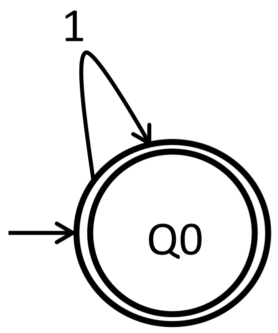
• Q is a finite set called the ***states.***	Q = {Q0, Q1, Q2}
• Σ is a finite set called the ***alphabet***.	$\Sigma = \{0,1\}$
• $\delta: Q \times \Sigma \mapsto Q$ is the ***transition function*** *(unique)*.	$(Q0,0) \mapsto Q1;\ (Q0,1) \mapsto Q0, \ldots\ldots, (Q2,1) \mapsto Q0$
• $q_0 \in Q$ is the ***start state***.	$q_0 = Q0$
• $F \subseteq Q$ are the ***final/accepting states***.	$F = Q2$

Constructing DFA for a language

Examples: $\boldsymbol{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, **L(M)={$\omega$|$\omega$ has an even number of 0's}**

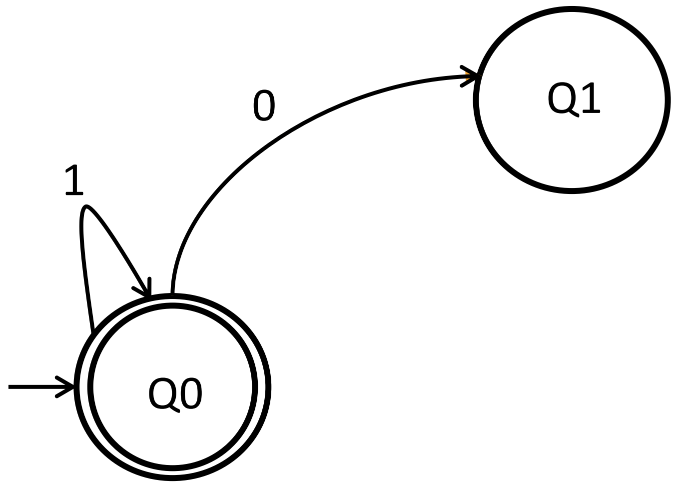
Constructing DFA for a language

Examples: $\boldsymbol{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, **L(M)={$\omega|\omega$ has an even number of 0's}**



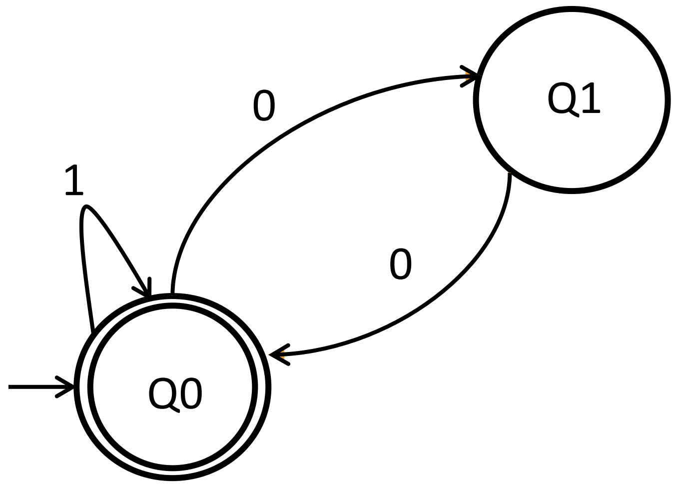
Constructing DFA for a language

Examples: $\boldsymbol{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, **L(M)={$\boldsymbol{\omega} | \boldsymbol{\omega}$ has an even number of 0’s}**



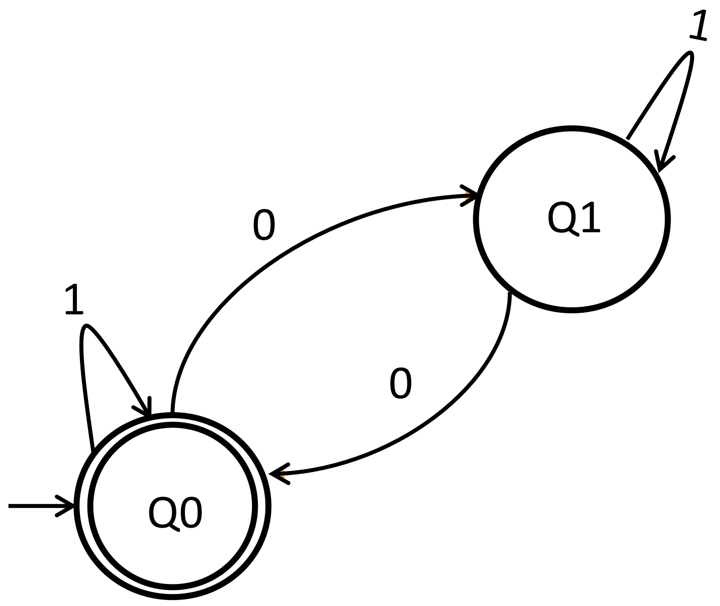
Constructing DFA for a language

Examples: $\mathbf{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, **L(M)={$\boldsymbol{\omega}|\boldsymbol{\omega}$ has an even number of 0's}**



Constructing DFA for a language

Examples: $\mathbf{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, **L(M)={$\omega$|$\omega$ has an even number of 0's}**



	0	**1**
Q0	**Q1**	**Q0**
Q1	**Q0**	**Q1**

Constructing DFA for a language

Examples: $\boldsymbol{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, **L(M)={$\boldsymbol{\omega}|\boldsymbol{\omega}$ is divisible by 3}**

Any input string would leave three remainders: $0, 1$ or 2.

Constructing DFA for a language

Examples: $\boldsymbol{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, **L(M)={$\boldsymbol{\omega}|\boldsymbol{\omega}$ is divisible by 3}**

Any input string would leave three remainders: $0, 1$ or 2.

Intuition: Let ω be any substring of the input string divisible by 3, i.e. $\omega = 0(mod\ 3)$

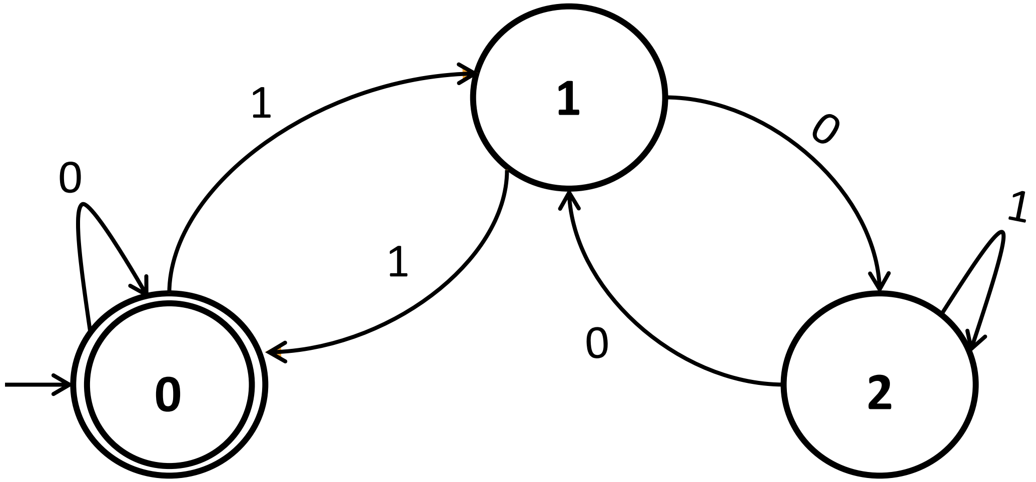
$$\omega\ 0 = 2 \times value\ (\omega) = 0\ (mod\ 3)$$
$$\omega\ 1 = 2 \times value(\omega) + 1 = 1(mod\ 3)$$
$$\omega\ 10 = 2 \times value(\omega 1) = 2(mod\ 3)$$
$$\omega\ 11 = 2 \times value(\omega 1) + 1 = 0(mod\ 3)$$

…. And so on

- The DFA will have three states, each corresponding to the remainder of $value(\omega)/3$.
- The final state $= 0(mod\ 3)$ – the string ω is accepted if after reading it, the DFA ends in this state.

Constructing DFA for a language

Examples: $\boldsymbol{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, **L(M)={$\boldsymbol{\omega}|\boldsymbol{\omega}$ is divisible by 3}**



Any input string would either leave remainders 0, 1 or 2.

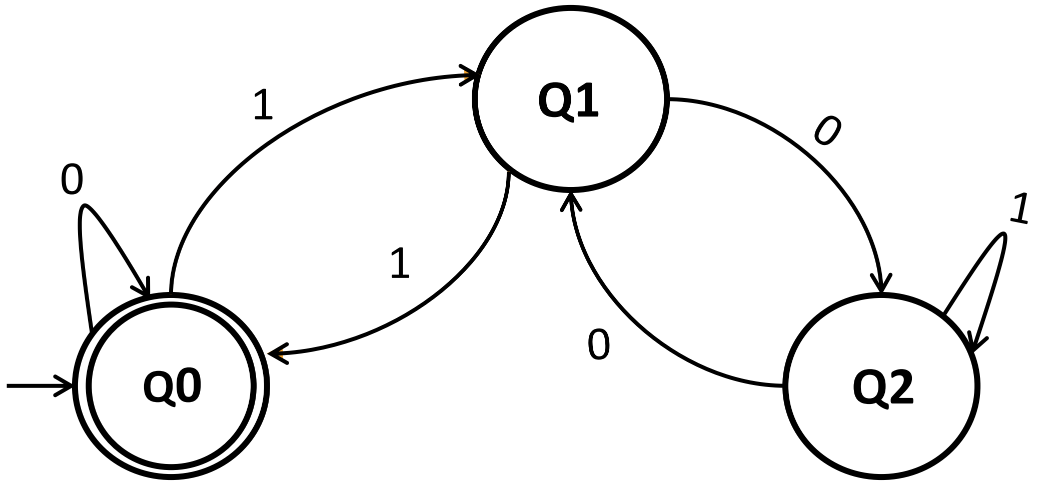
Intuition: Let ω be any substring of the input string divisible by 3, i.e. $\omega = 0(mod\ 3)$

$$\omega\ 0 = 2 \times value\ (\omega) = 0\ (\text{mod } 3)$$
$$\omega\ 1 = 2 \times value(\omega) + 1 = 1(mod\ 3)$$
$$\omega\ 10 = 2 \times value(\omega 1) = 2(mod\ 3)$$
$$\omega\ 11 = 2 \times value(\omega 1) + 1 = 0(mod\ 3)$$

…. And so on

Constructing DFA for a language

Examples: $\boldsymbol{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, **L(M)={$\boldsymbol{\omega}$|$\boldsymbol{\omega}$ is divisible by 3}**



	0	**1**
Q0	**Q0**	**Q1**
Q1	**Q2**	**Q0**
Q2	**Q1**	**Q2**

Constructing DFA for a language

Examples: $\boldsymbol{\Sigma} = \{\boldsymbol{0}, \boldsymbol{1}\}$, **L(M)={$\boldsymbol{\omega}|\boldsymbol{\omega}$ is NOT divisible by 3}**

Constructing DFA for a language

Examples: $\boldsymbol{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, **L(M)={$\boldsymbol{\omega}$|$\boldsymbol{\omega}$ is NOT divisible by 3}**

Intuition - Construct a **Toggled DFA:** Toggle the final states and the non-final states!

Constructing DFA for a language

Examples: $\boldsymbol{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, **L(M)={$\boldsymbol{\omega}$|$\boldsymbol{\omega}$ is NOT divisible by 3}**

Intuition - Construct a **Toggled DFA:** Toggle the final states and the non-final states!

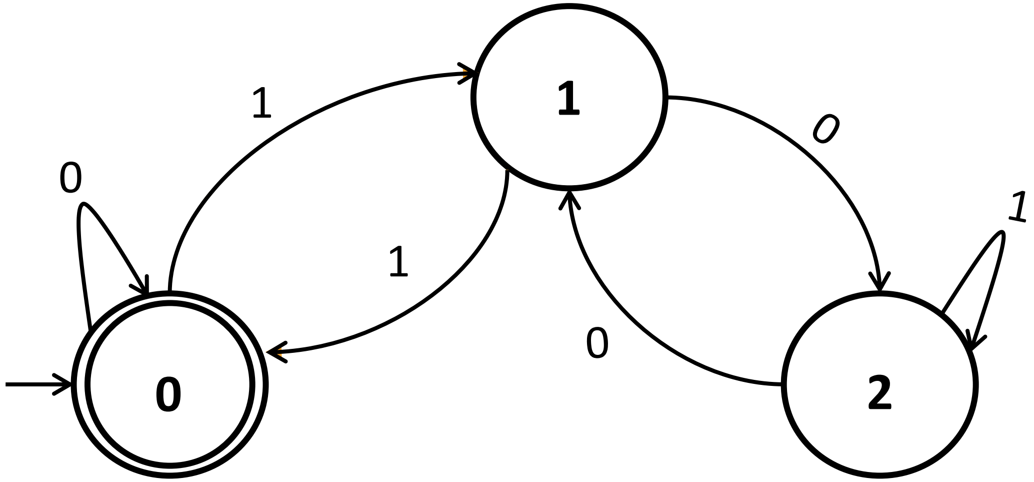
In fact if any DFA accepts $\boldsymbol{L}$, the toggled DFA accepts $\overline{\boldsymbol{L}}$, the complement of $\boldsymbol{L}$

Constructing DFA for a language

Examples: $\mathbf{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, **L(M)={$\boldsymbol{\omega}$|$\boldsymbol{\omega}$ is NOT divisible by 3}**

Intuition - Construct a **Toggled DFA:** Toggle the final states and the non-final states!

In fact if any DFA accepts $\boldsymbol{L}$, the toggled DFA accepts $\overline{\boldsymbol{L}}$, the complement of $\boldsymbol{L}$

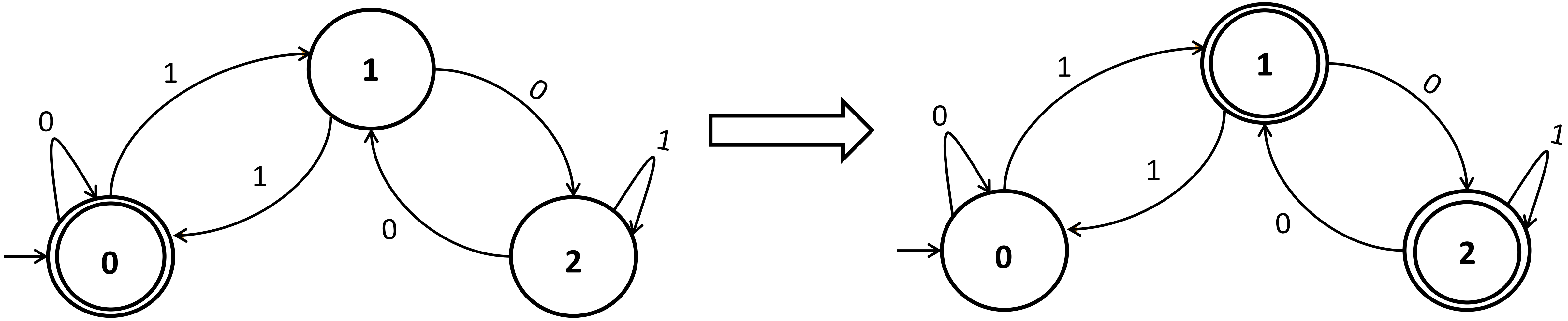


Constructing DFA for a language

Examples: $\mathbf{\Sigma} = \{\mathbf{0}, \mathbf{1}\}$, **L(M)={$\boldsymbol{\omega}$|$\boldsymbol{\omega}$ is NOT divisible by 3}**

Intuition - Construct a **Toggled DFA:** Toggle the final states and the non-final states!

In fact if any DFA accepts $\boldsymbol{L}$, the toggled DFA accepts $\overline{\boldsymbol{L}}$, the complement of $\boldsymbol{L}$



Non-deterministic Finite Automata (NFA)

Characteristics of DFA : (i) Single start state (ii) Unique transitions (iii) Zero or more final states

Non-deterministic Finite Automata (NFA)

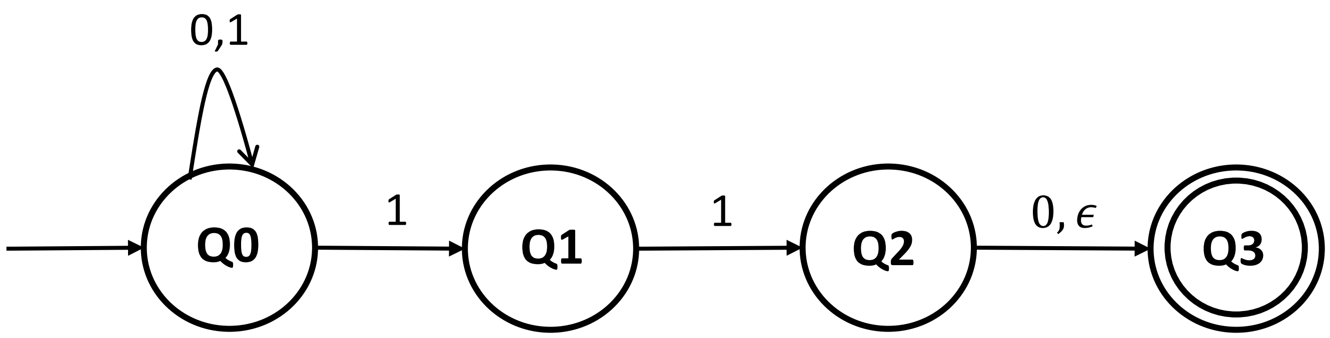
Characteristics of DFA : (i) Single start state (ii) Unique transitions (iii) Zero or more final states

Characteristics of NFA : (i) Single start state (ii) Zero or more final states

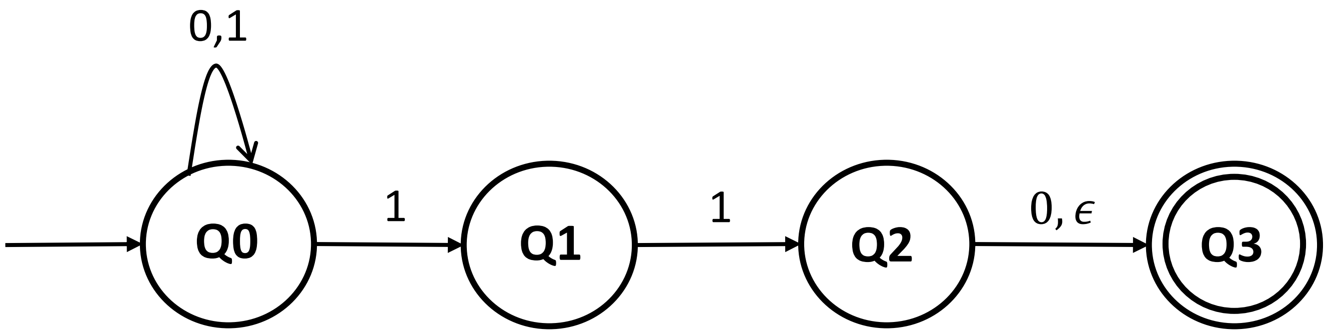
(iii) Multiple transitions are possible on the same input for a state

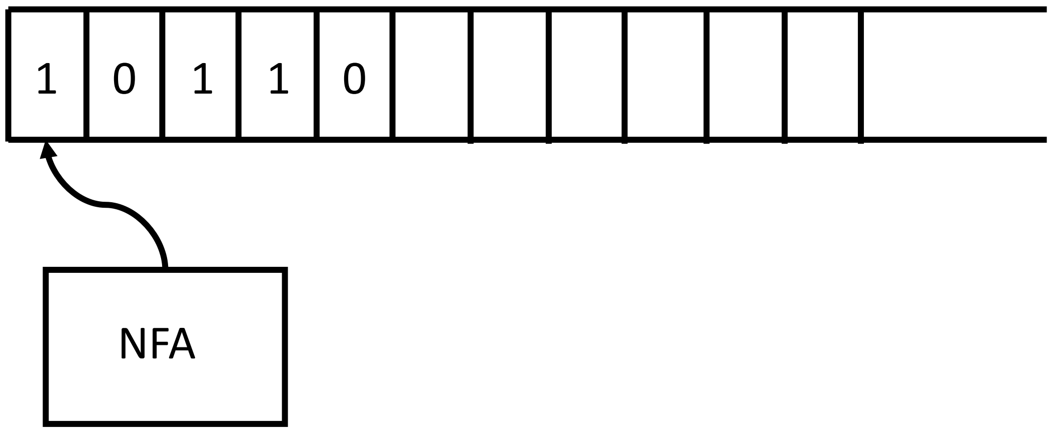
(iv) Some transitions might be missing

(v) ϵ - transitions



Non-deterministic Finite Automata (NFA)



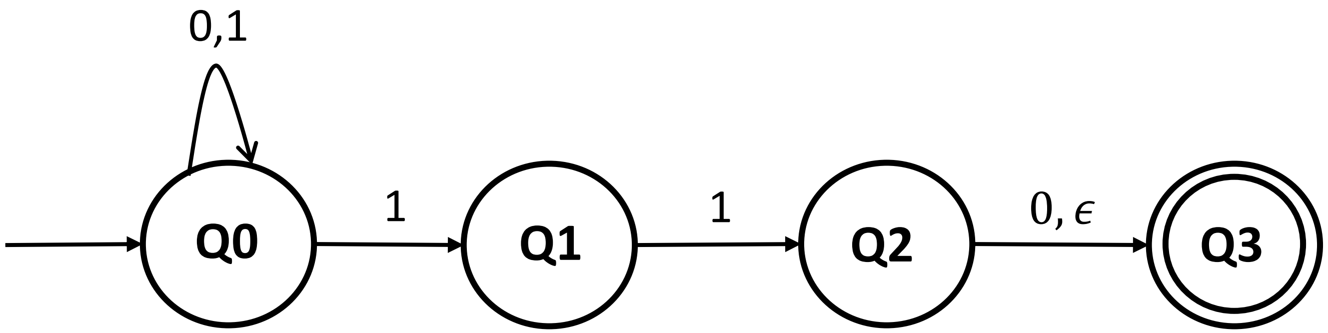


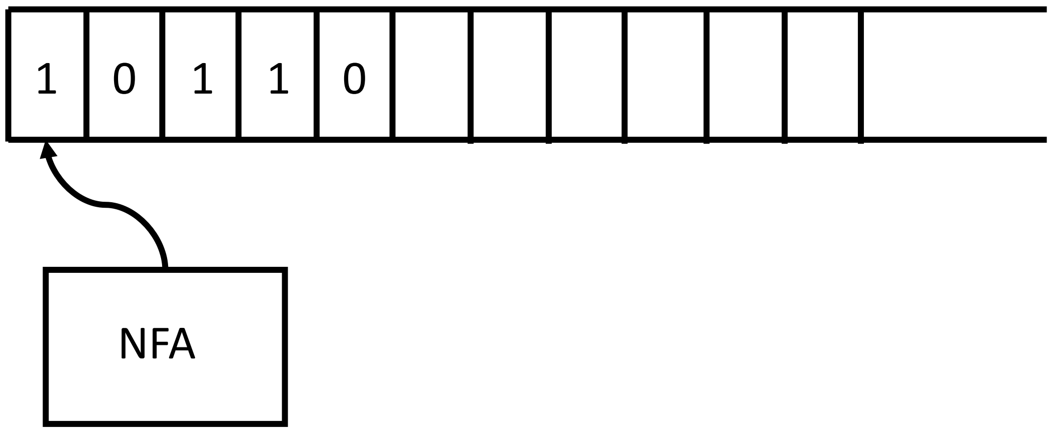
Run 1: $Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q0 \xrightarrow{1} Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q0$ (**REJECT**)

Run 2: $Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q0 \xrightarrow{1} Q1 \xrightarrow{1} Q2 \xrightarrow{0} Q3$ (**ACCEPT**)

Multiple **runs** per input is possible

Non-deterministic Finite Automata (NFA)





Run 1: $Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q0 \xrightarrow{1} Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q0$ (**REJECT**)

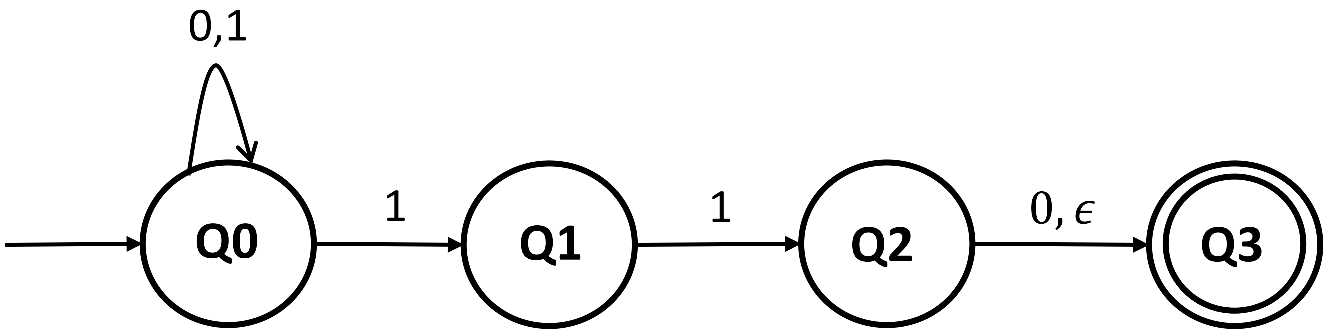
Run 2: $Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q0 \xrightarrow{1} Q1 \xrightarrow{1} Q2 \xrightarrow{0} Q3$ (**ACCEPT**)

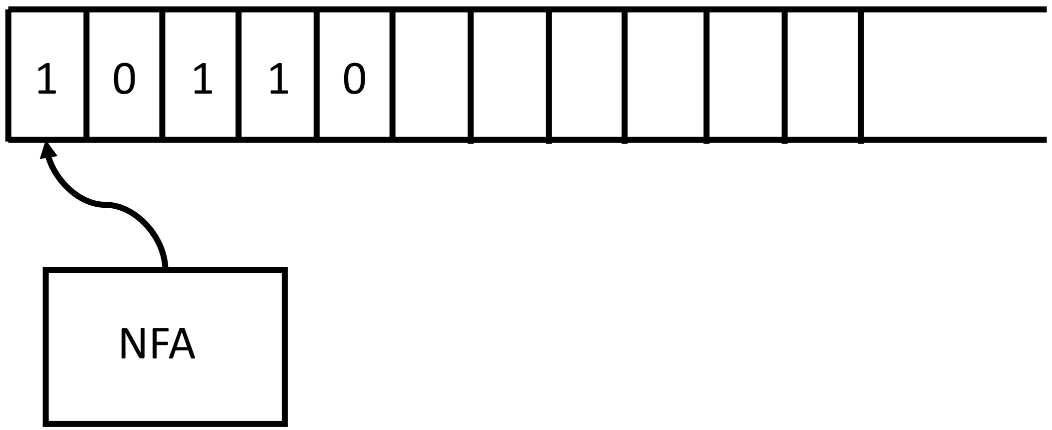
Run 3: $Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q0 \xrightarrow{1} Q0 \xrightarrow{1} Q1 \xrightarrow{0}$ **CRASH**

Run 4: $Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q0 \xrightarrow{1} Q1 \xrightarrow{1} Q2 \xrightarrow{\epsilon} Q3 \xrightarrow{0}$ **CRASH**

CRASH is a Rejecting Run

Non-deterministic Finite Automata (NFA)





Run 1: $Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q0 \xrightarrow{1} Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q0$ (**REJECT**)

Run 2: $Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q0 \xrightarrow{1} Q1 \xrightarrow{1} Q2 \xrightarrow{0} Q3$ (**ACCEPT**)

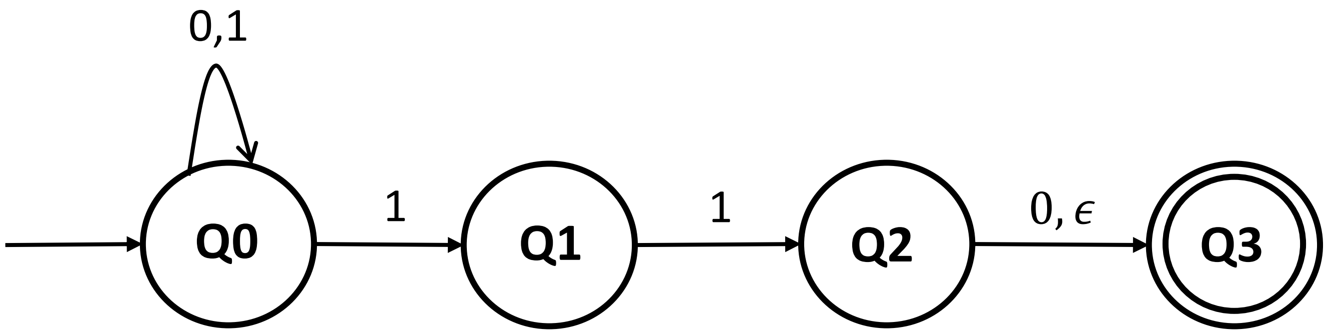
Run 3: $Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q0 \xrightarrow{1} Q0 \xrightarrow{1} Q1 \xrightarrow{0}$ CRASH **(REJECT)**

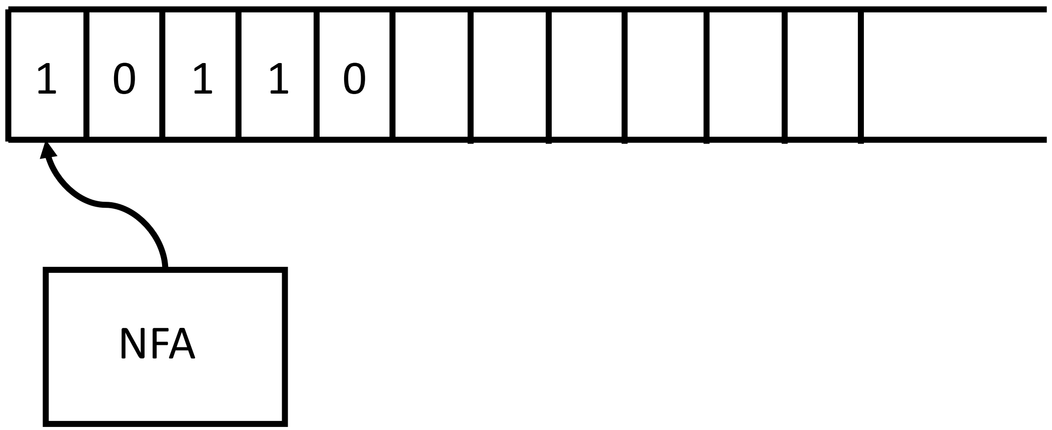
Run 4: $Q0 \xrightarrow{1} Q0 \xrightarrow{0} Q0 \xrightarrow{1} Q1 \xrightarrow{1} Q2 \xrightarrow{\epsilon} Q3 \xrightarrow{0}$ CRASH **(REJECT)**

The NFA “accepts” an input string, if it at **least one *run* ends up in the final state**. **(Accepting Run)**

The NFA “rejects” an input string, if there are **no runs that end up in a final state**. **(Rejecting Run)**

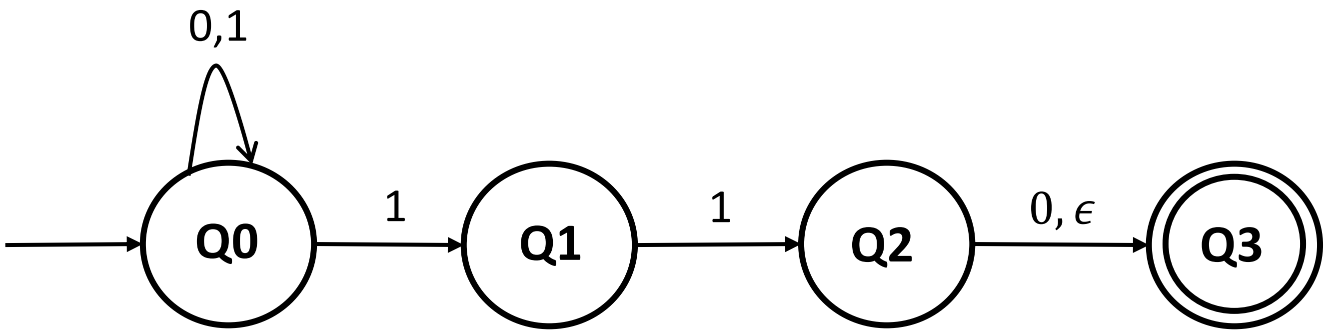
Non-deterministic Finite Automata (NFA)

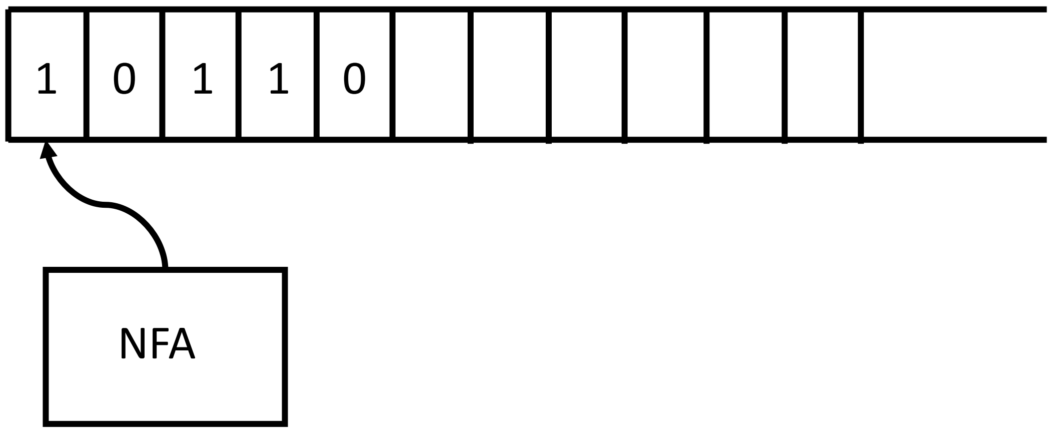




	0	**1**	ϵ
Q0	**Q0**	**Q0, Q1**	
Q1		**Q2**	
Q2	**Q3**		**Q3**
Q3			

Non-deterministic Finite Automata (NFA)





Formally, a finite automaton M is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

- Q is a finite set called the ***states.***
- Σ is a finite set called the ***alphabet***.
- $\delta: Q \times \Sigma \mapsto P(Q)$ is the ***transition function***. $P(Q)$ is the power set of Q
- $q_0 \in Q$ is the ***start state***.
- $F \subseteq Q$ is the set of ***final/accepting states***.

	0	**1**	ϵ
Q0	**Q0**	**Q0, Q1**	
Q1		**Q2**	
Q2	**Q3**		**Q3**
Q3			

NFA vs DFA

- Are NFAs more powerful than DFAs?
- Intuitively, non-determinism seems to be adding more “power”.

NFA vs DFA

- Are NFAs more powerful than DFAs?
- Intuitively, non-determinism seems to be adding more “power”.
- Let L_1 be the language accepted by NFAs and L_2 be the language accepted by DFAs
- Is $L_2 \subseteq L_1$?

NFA vs DFA

- Are NFAs more powerful than DFAs?
- Intuitively, non-determinism seems to be adding more "power".
- Let L_1 be the language accepted by NFAs and L_2 be the language accepted by DFAs
- Is $L_2 \subseteq L_1$? Clearly true, because a DFA is just a special case of an NFA.

NFA vs DFA

- Are NFAs more powerful than DFAs?
- Intuitively, non-determinism seems to be adding more "power".
- Let L_1 be the language accepted by NFAs and L_2 be the language accepted by DFAs
- Is $L_2 \subseteq L_1$? Clearly true, because a DFA is just a special case of an NFA.
- Surprisingly, what we will show next is that $\boldsymbol{L_1 \subseteq L_2}$!
- That is, **given an NFA, we can convert it to a DFA that accepts the same language**.
- Such a DFA is called a "**Remembering DFA**".

Thus, DFAs and NFAs are completely equivalent and $\boldsymbol{L_1 = L_2}$!

Converting an NFA to a DFA

Intuitive idea for the construction of a Remembering DFA from an NFA:

- Let R be the Remembering DFA corresponding to an NFA N.
- R on an input enters a state that is labelled by all possible states that N can enter on that input.
- Note that this “trims away” the non-determinism of the NFA N without “losing” the language it accepts.
- Also note that if N has k states, then R has at most 2^k states. Why?

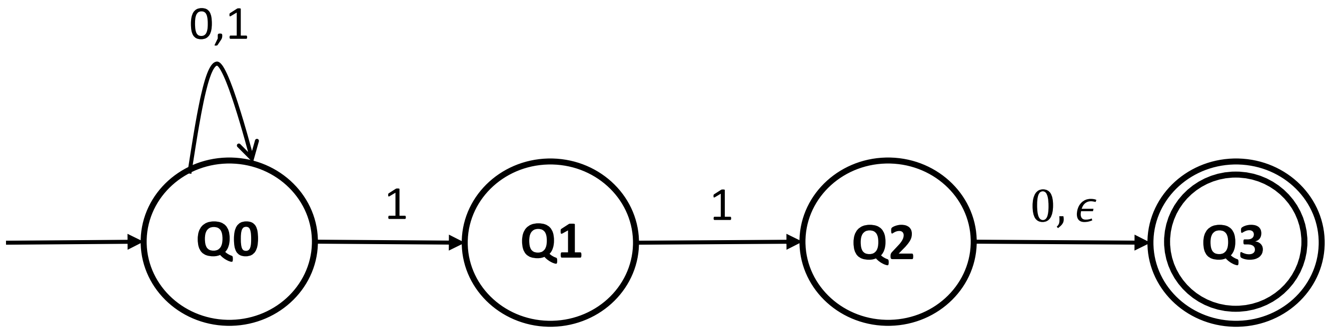
Converting an NFA to a DFA

Intuitive idea for the construction of a Remembering DFA from an NFA:

- Let R be the Remembering DFA corresponding to an NFA N.
- R on an input enters a state that is labelled by all possible states that N can enter on that input.
- Note that this “trims away” the non-determinism of the NFA N without “losing” the language it accepts.
- Also note that if N has k states, then R has at most 2^k states. Why?
- Any label in the Remembering DFA is a subset of $\{Q_0,\ Q_1,\ Q_2,\ldots\ldots,Q_{k-1}\}$, where Q_i = State of the NFA.
- There are at most 2^k labels for the DFA.

Converting an NFA to a DFA

- R on an input enters a state that is labelled by all possible states that N can enter on that input.

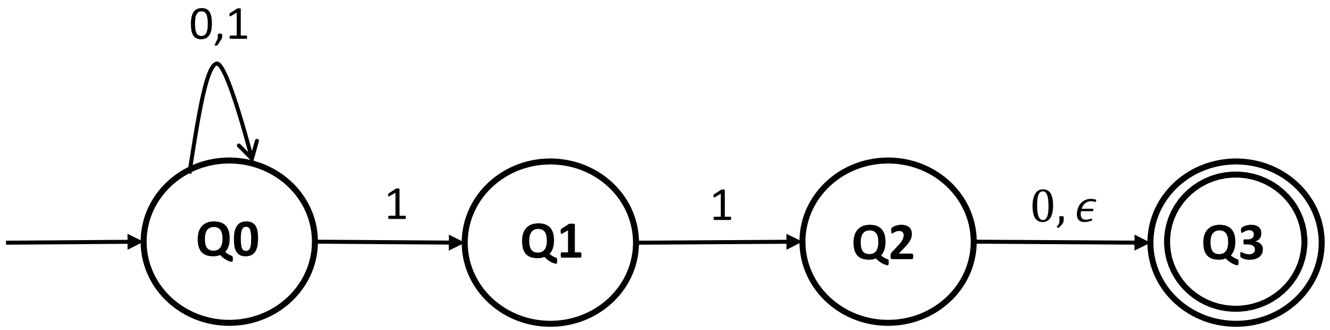


NFA N

	0	**1**	$\boldsymbol{\epsilon}$
Q0	**Q0**	**Q0, Q1**	
Q1		**Q2**	
Q2	**Q3**		**Q3**
Q3			

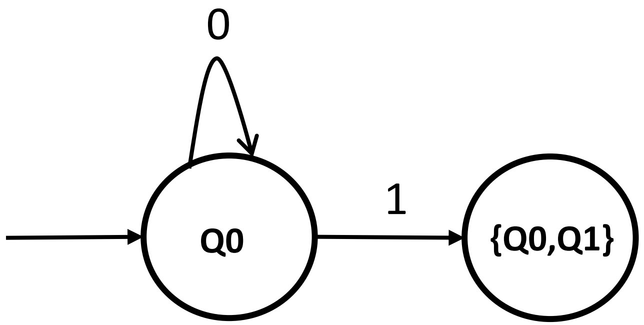
Converting an NFA to a DFA

- R on an input enters a state that is labelled by all possible states that N can enter on that input.



NFA N

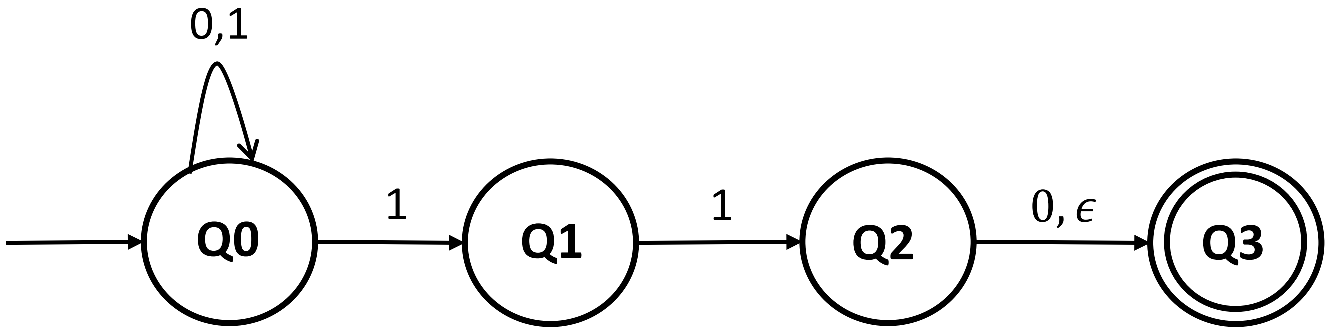
	0	1	ϵ
Q0	**Q0**	**Q0, Q1**	
Q1		**Q2**	
Q2	**Q3**		**Q3**
Q3			



Remembering DFA R

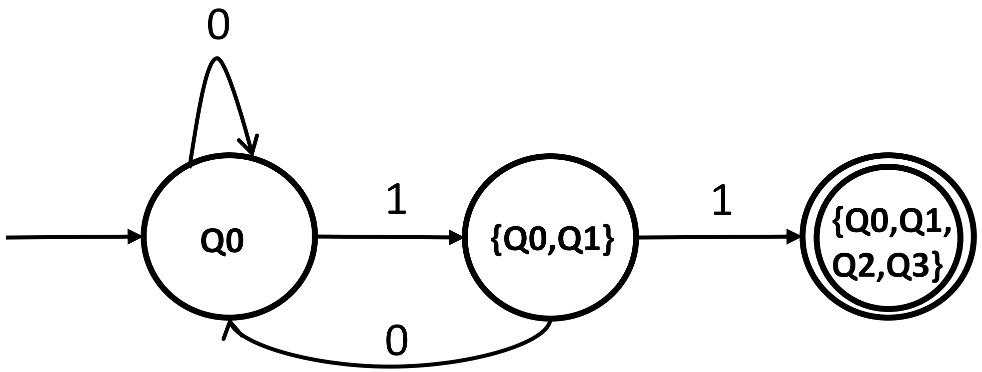
Converting an NFA to a DFA

- R on an input enters a state that is labelled by all possible states that N can enter on that input.



NFA N

	0	1	ϵ
Q0	**Q0**	**Q0, Q1**	
Q1		**Q2**	
Q2	**Q3**		**Q3**
Q3			

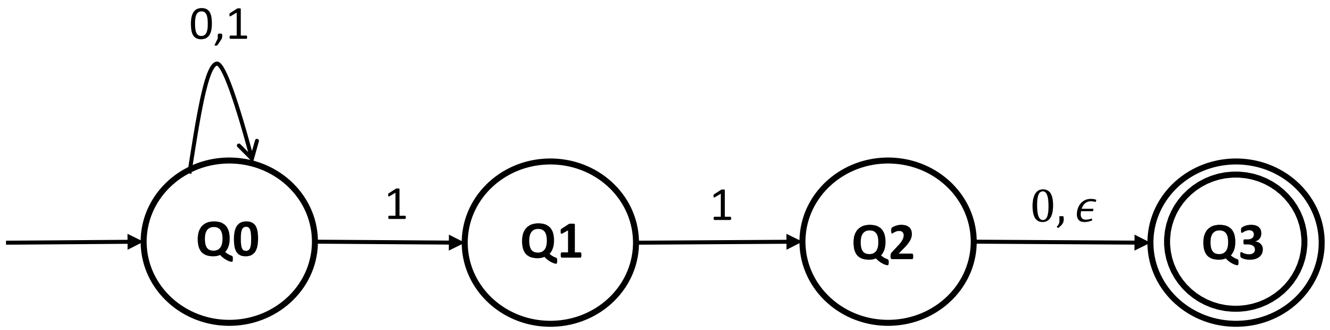


Remembering DFA R

Any state of R that contains in its label, an accepting state of N is an accepting state of R.

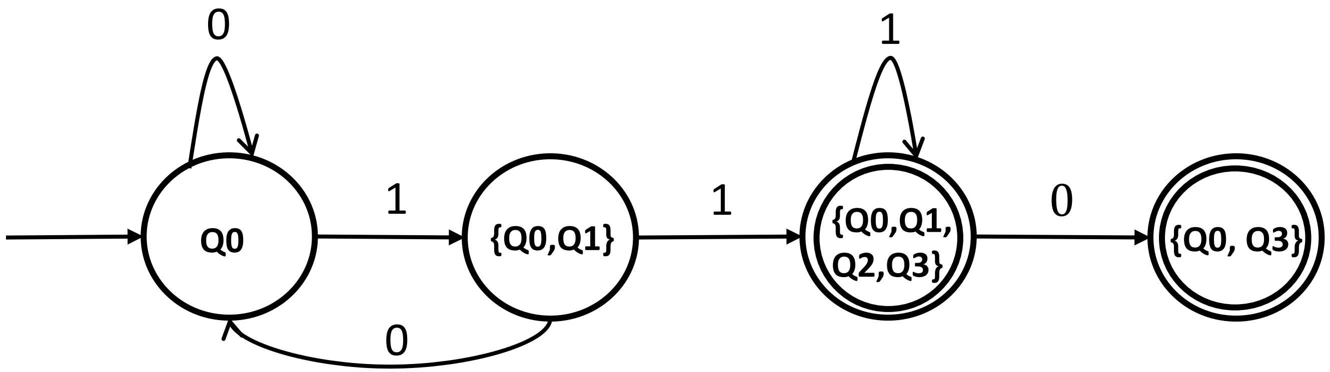
Converting an NFA to a DFA

- R on an input enters a state that is labelled by all possible states that N can enter on that input.



NFA N

	0	**1**	ϵ
Q0	**Q0**	**Q0, Q1**	
Q1		**Q2**	
Q2	**Q3**		**Q3**
Q3			

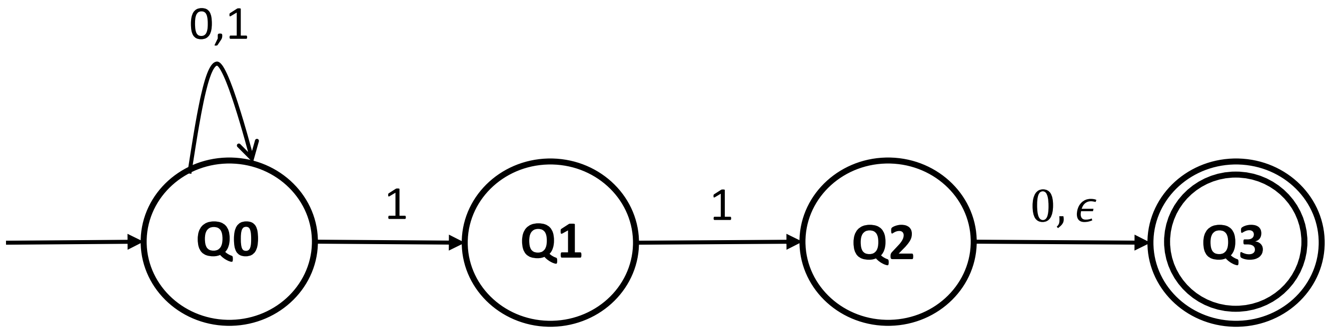


Remembering DFA R

Any state of R that contains in its label, an accepting state of N is an accepting state of R.

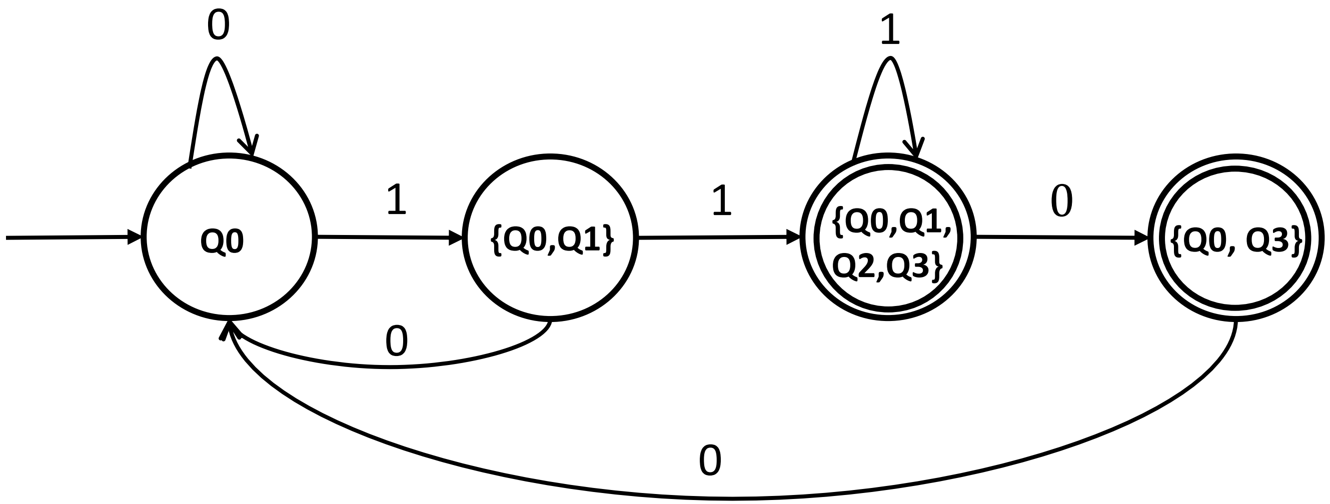
Converting an NFA to a DFA

- M_2 on an input enters a state that is labelled by all possible states that M_1 can enter on that input.



NFA N

	0	**1**	ϵ
Q0	**Q0**	**Q0, Q1**	
Q1		**Q2**	
Q2	**Q3**		**Q3**
Q3			

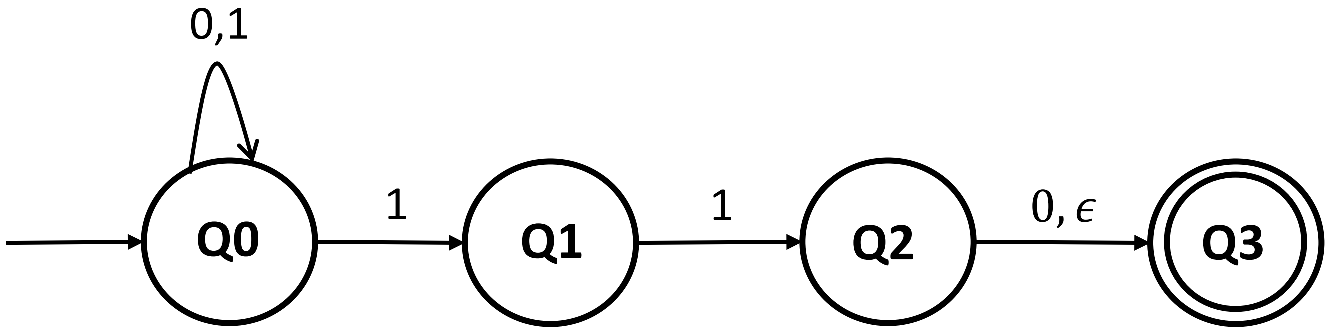


Remembering DFA R

Any state of R that contains in its label, an accepting state of N is an accepting state of R.

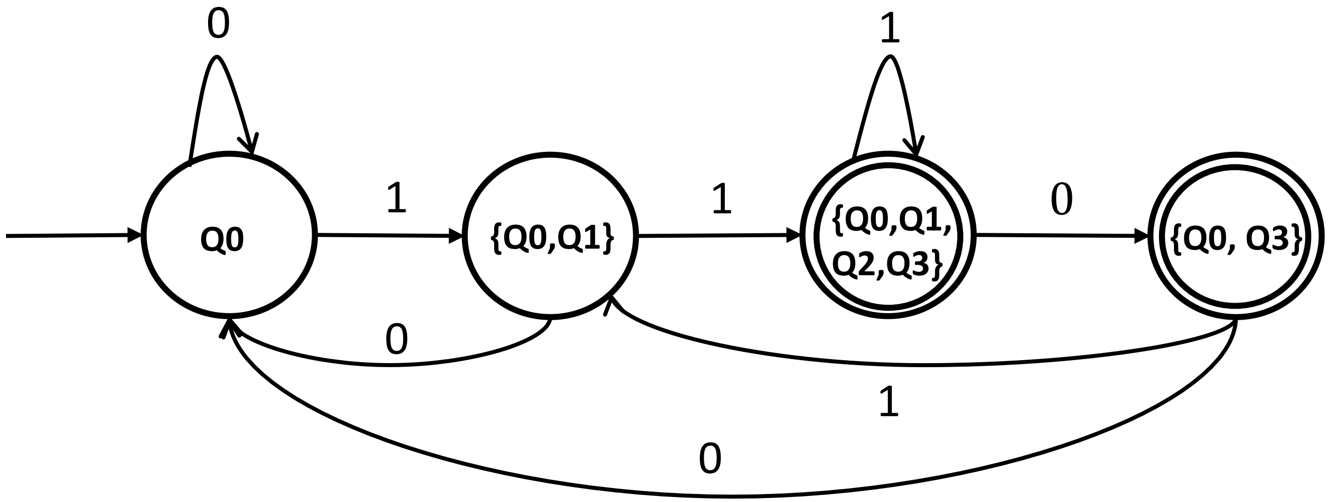
Converting an NFA to a DFA

- M_2 on an input enters a state that is labelled by all possible states that M_1 can enter on that input.



NFA N

	0	**1**	ϵ
Q0	**Q0**	**Q0, Q1**	
Q1		**Q2**	
Q2	**Q3**		**Q3**
Q3			

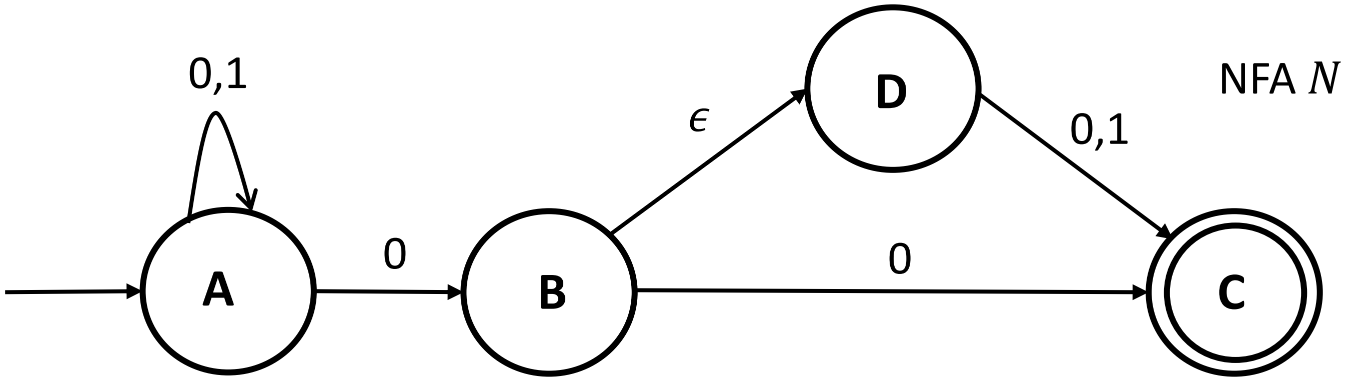


Remembering DFA R

Any state of R that contains in its label, an accepting state of N is an accepting state of R.

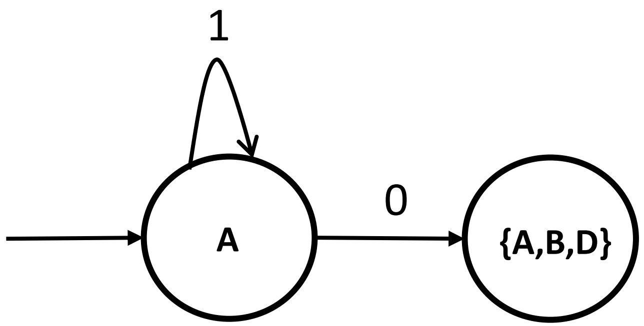
Converting an NFA to a DFA

- M_2 on an input enters a state that is labelled by all possible states that M_1 can enter on that input.



NFA N

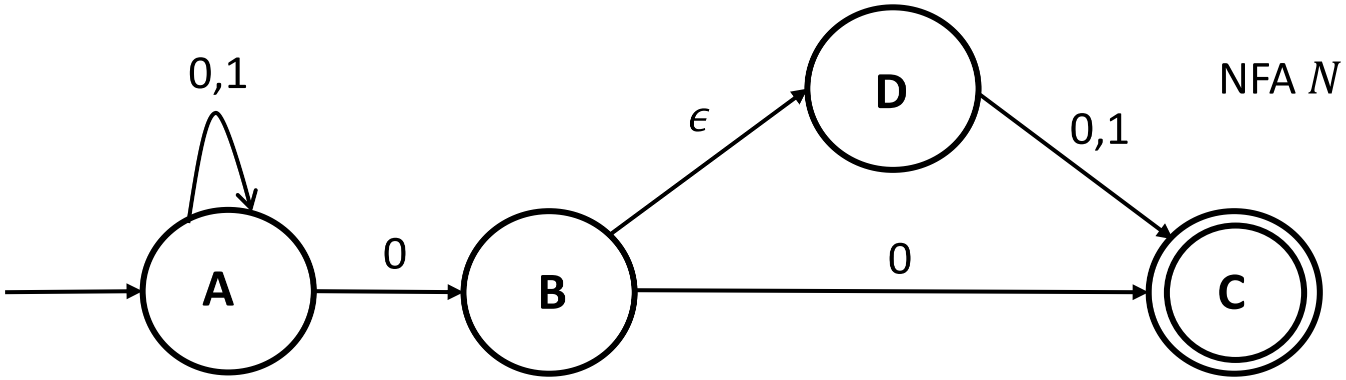
	0	1	ϵ
A	A, B	A	
B	C		D
C			
D	C	C	



Remembering DFA R

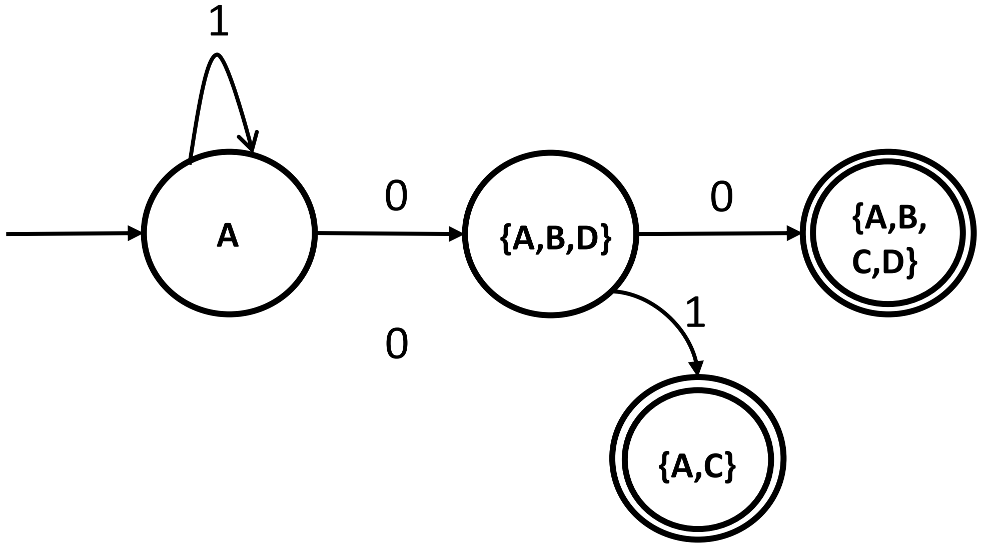
Converting an NFA to a DFA

- M_2 on an input enters a state that is labelled by all possible states that M_1 can enter on that input.



NFA N

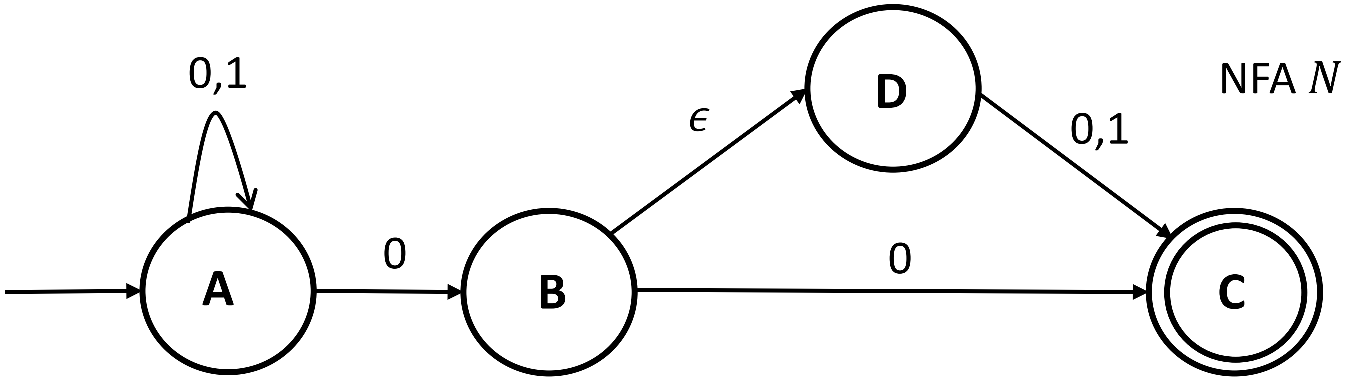
	0	1	ϵ
A	**A, B**	**A**	
B	**C**		**D**
C			
D	**C**	**C**	



Remembering DFA R

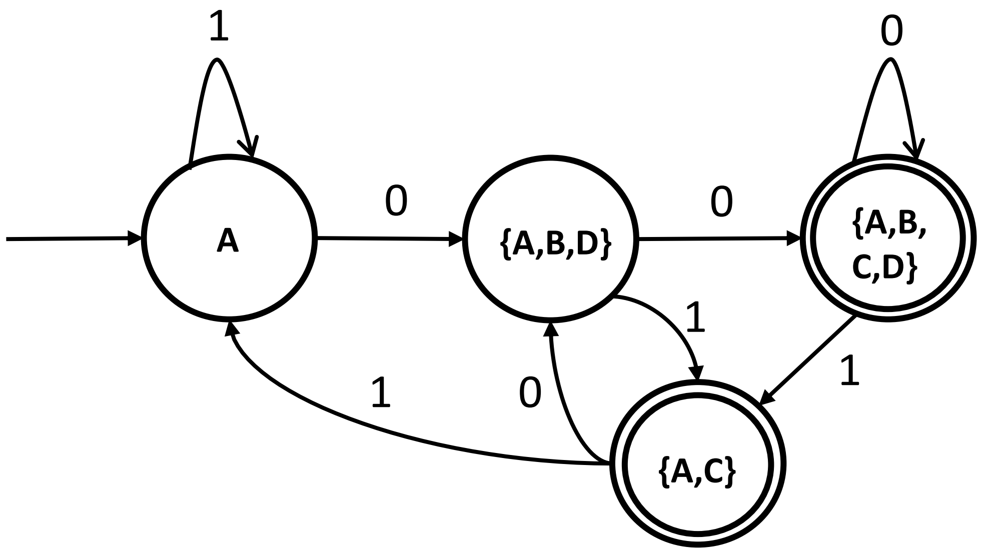
Converting an NFA to a DFA

- M_2 on an input enters a state that is labelled by all possible states that M_1 can enter on that input.



NFA N

	0	1	ϵ
A	A, B	A	
B	C		D
C			
D	C	C	



Remembering DFA R

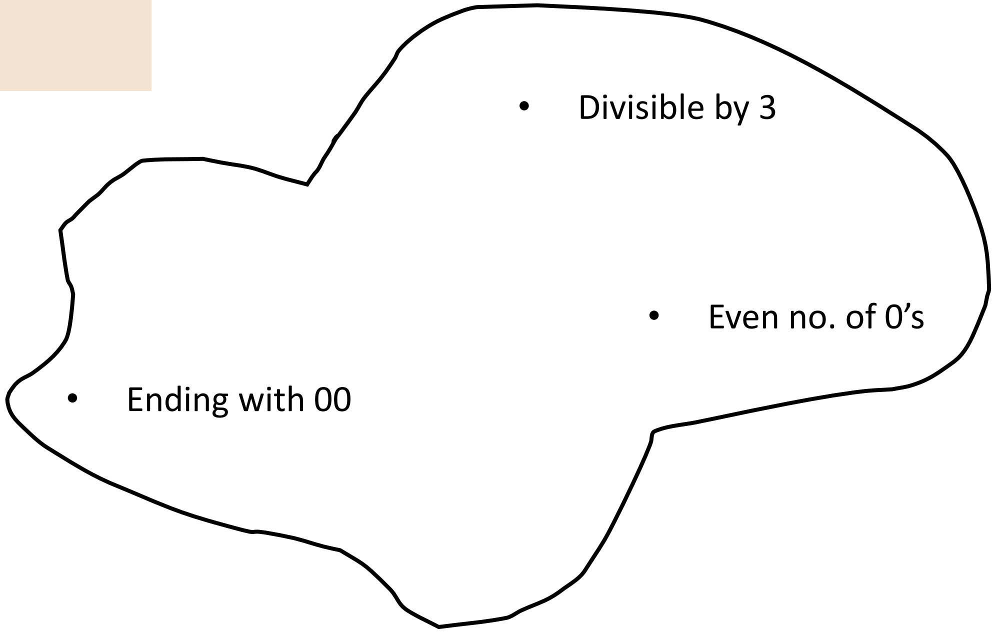
Regular Languages

A language is called a **Regular Language** if there exists some finite automata recognizing it.

If M be a finite automaton (DFA/NFA) and,

$$L(M) = \{\omega | \omega \text{ is accepted by } M\}$$

$\boldsymbol{L(M)}$ **is regular.**



Set of all regular Languages

Regular Languages

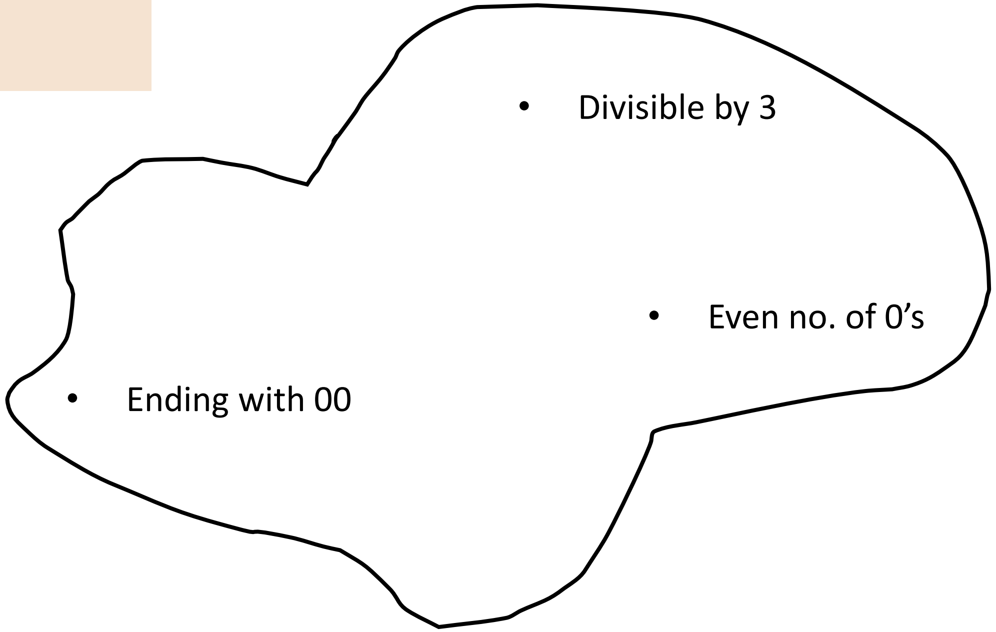
A language is called a **Regular Language** if there exists some finite automata recognizing it.

If M be a finite automaton (DFA/NFA) and,

$$L(M) = \{\omega | \omega \text{ is accepted by } M\}$$

$L(M)$ is regular.

- Any language has associated with it, a set of operations that can be performed on it.
- These operations help us to understand the properties of that language, e.g. closure properties
- For regular languages, this will help us prove that certain languages are non-regular and hence we cannot hope to design a finite automaton for them



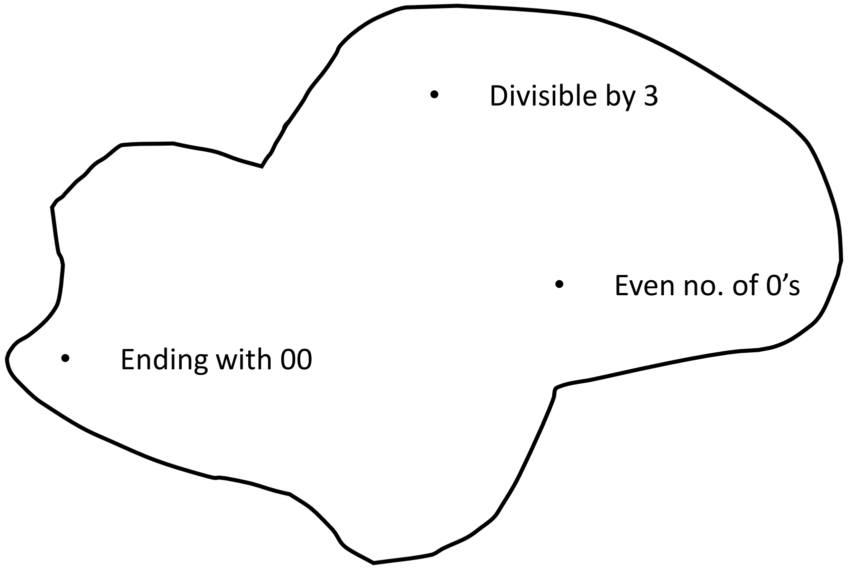
Set of all regular Languages

Regular Languages

Regular Operations:

Let L_1 and L_2 be languages. The following are the *regular operations*:

- **Union:** $L_1 \cup L_2 = \{x | x \in L_1 \text{ or } x \in L_2\}$
- **Concatenation:** $L_1 . L_2 = \{xy | x \in L_1 \text{ and } y \in L_2\}$
- **Star:** $L_1^* = \{x_1 x_2 \cdots x_k |\ k \geq 0 \text{ and each } x_i \in L_1\}$



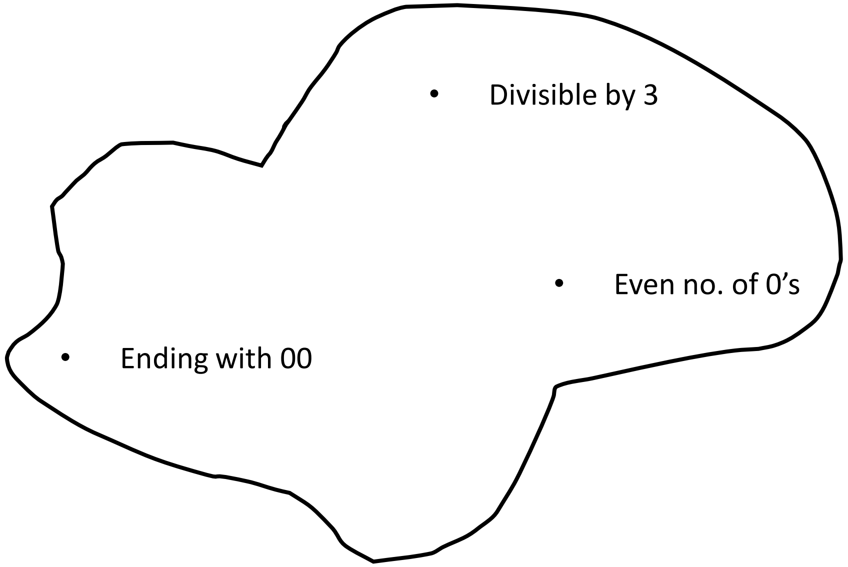
Set of all regular Languages

Regular Languages

Regular Operations:

Let L_1 and L_2 be languages. The following are the *regular operations*:

- **Union:** $L_1 \cup L_2 = \{x | x \in L_1 \text{ or } x \in L_2\}$
- **Concatenation:** $L_1 . L_2 = \{xy | x \in L_1 \text{ and } y \in L_2\}$
- **Star:** $L_1^* = \{x_1 x_2 \cdots x_k |\ k \geq 0 \text{ and each } x_i \in L_1\}$



Set of all regular Languages

Star operation: It is an unary operation (unlike the other two) and involves putting together *any number of strings in L_1 together to obtain a new string.*

Note: Any number of strings includes "0" as a possibility and so the empty string ϵ is a member of L_1^***.**

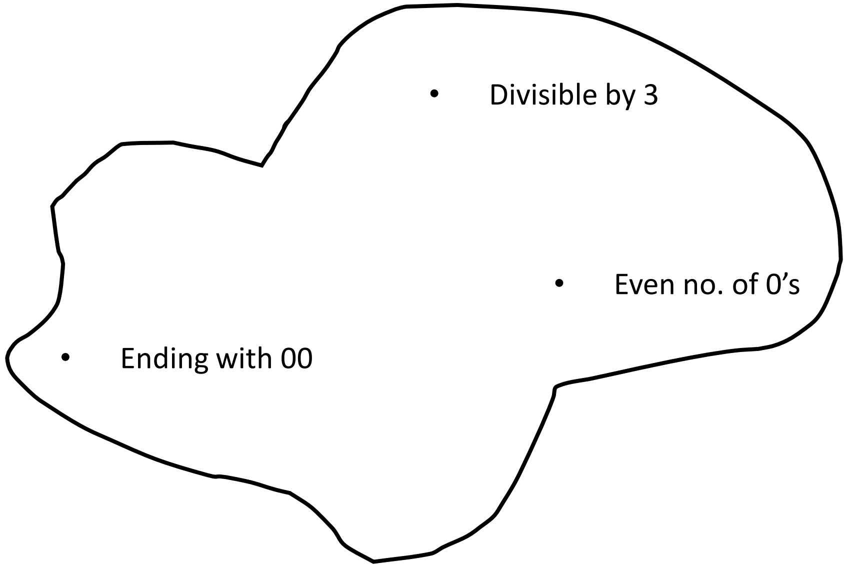
If $\Sigma = \{a\}$, $\Sigma^* = \{\epsilon, a, aa, aaa, \ldots \ldots\}$; If $\Sigma = \{\Phi\}, \Sigma^* = \{\epsilon\}$

Regular Languages

Regular Operations:

Let L_1 and L_2 be languages. The following are the *regular operations*:

- **Union:** $L_1 \cup L_2 = \{x | x \in L_1 \text{ or } x \in L_2\}$
- **Concatenation:** $L_1 . L_2 = \{xy | x \in L_1 \text{ and } y \in L_2\}$
- **Star:** $L_1^* = \{x_1 x_2 \cdots x_k |\, k \geq 0 \text{ and each } x_i \in L_1\}$



Set of all regular Languages

Star operation: It is an unary operation (unlike the other two) and involves putting together *any number of strings in L_1 together to obtain a new string.*

Note: Any number of strings includes "0" as a possibility and so the empty string ϵ is a member of L_1^***.**

If $\Sigma = \{0,1\}$, we have that $\Sigma^* = \{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots \ldots\}$

Regular Languages

Regular Operations: Let L_1 and L_2 be languages.

- **Union:** $L_1 \cup L_2 = \{x | x \in L_1 \text{ or } x \in L_2\}$
- **Concatenation:** $L_1 . L_2 = \{xy | x \in L_1 \text{ and } y \in L_2\}$
- **Star:** $L_1^* = \{x_1 x_2 \cdots x_k |\ k \geq 0 \text{ and each } x_i \in L_1\}$

Example: Let the alphabet $\Sigma = \{a, b, \cdots, z\}$. If $L_1 = \{\textit{social, economic}\}$ and $L_2 = \{\textit{justice, reform}\}$, then

- $L_1 \cup L_2 = \{\textit{social, economic, justice, reform}\}$

Regular Languages

Regular Operations: Let L_1 and L_2 be languages.

- **Union:** $L_1 \cup L_2 = \{x | x \in L_1 \text{ or } x \in L_2\}$
- **Concatenation:** $L_1 . L_2 = \{xy | x \in L_1 \text{ and } y \in L_2\}$
- **Star:** $L_1^* = \{x_1 x_2 \cdots x_k | \ k \geq 0 \text{ and each } x_i \in L_1\}$

Example: Let the alphabet $\Sigma = \{a, b, \cdots, z\}$. If $L_1 = \{$*social, economic*$\}$ and $L_2 = \{$*justice, reform*$\}$, then

- $L_1 \cup L_2 = \{$*social, economic, justice, reform*$\}$
- $L_1 . L_2 = \{$*socialjustice, socialreform, economicjustice, economicreform*$\}$

Regular Languages

Regular Operations: Let L_1 and L_2 be languages.

- **Union:** $L_1 \cup L_2 = \{x | x \in L_1$ or $x \in L_2\}$
- **Concatenation:** $L_1 . L_2 = \{xy | x \in L_1$ and $y \in L_2\}$
- **Star:** $L_1^* = \{x_1 x_2 \cdots x_k |\ k \geq 0$ and each $x_i \in L_1\}$

Example: Let the alphabet $\Sigma = \{a, b, \cdots, z\}$. If $L_1 = \{$*social, economic*$\}$ and $L_2 = \{$*justice, reform*$\}$, then

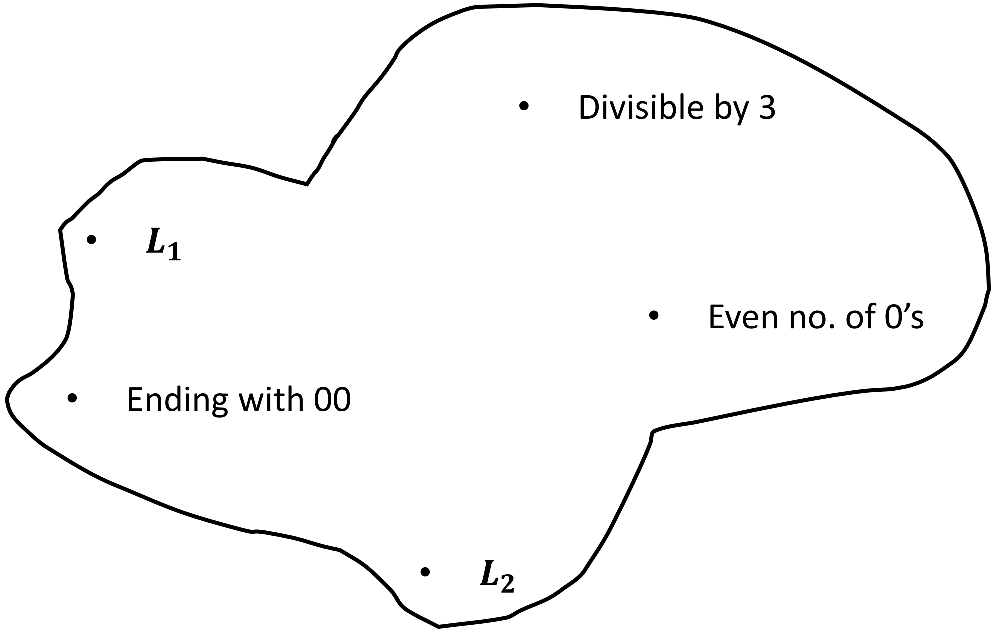
- $L_1 \cup L_2 = \{$*social, economic, justice, reform*$\}$
- $L_1 . L_2 = \{$*socialjustice, socialreform, economicjustice, economicreform*$\}$
- $L_1^* = \{\epsilon$, *social, economic, socialsocial, socialeconomic, economicsocial, economiceconomic, socialsocialsocial, socialsocialeconomic, socialeconomiceconomic,……*$\}$
- $L_2^* = \{\epsilon$, *justice, reform, justicejustice, justicereform, reformjustice, reformreform, justicejusticejustice, ……*$\}$

Closure of Regular Languages

We want to check whether the set of regular languages are **closed** under some operations.

What does this mean?

- We pick up points within the set of all regular languages (say L_1 and L_2)
- Perform *set operations* such as Union, concatenation, Star, intersection, reversal, complement etc on them.
- Observe whether the resulting language still belongs to the set of all regular languages.
- If so, we say, regular languages are **closed** under that operation.



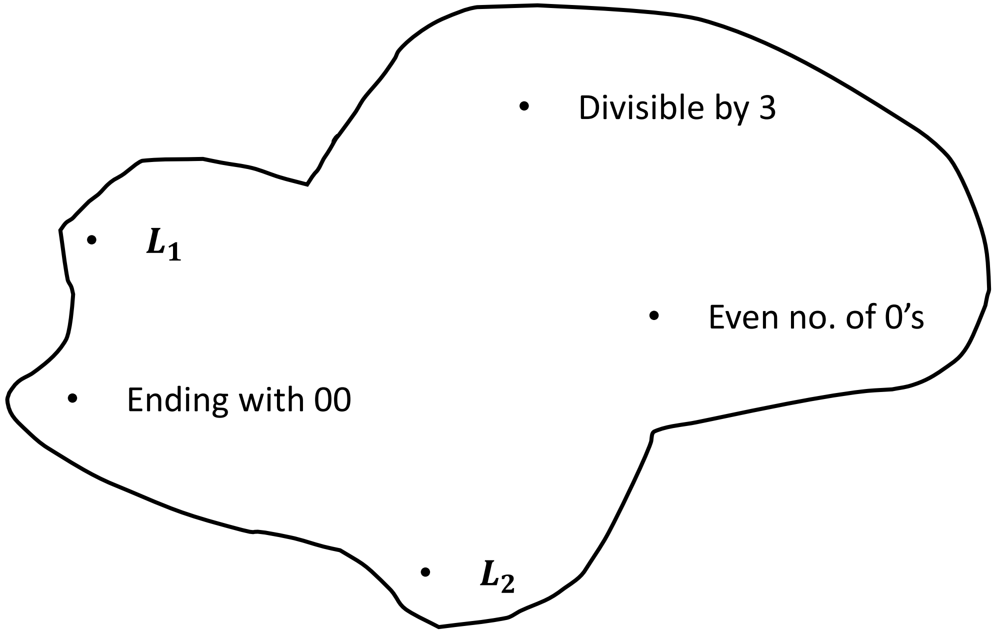
Set of all regular Languages

Closure of Regular Languages

We want to check whether the set of regular languages are **closed** under some operations.

What does this mean?

- We pick up points within the set of all regular languages (say L_1 and L_2)
- Perform *set operations* such as Union, concatenation, Star, intersection, reversal, complement etc on them.
- Observe whether the resulting language still belongs to the set of all regular languages.
- If so, we say, regular languages are **closed** under that operation.



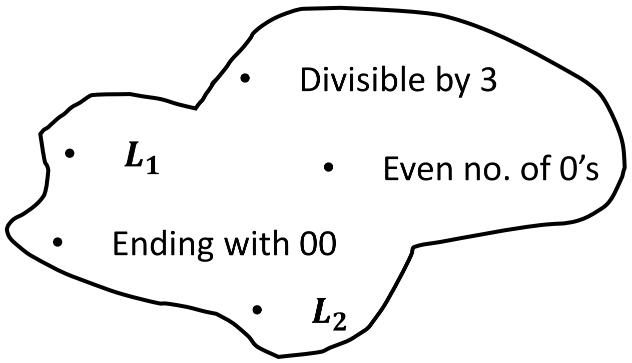
Set of all regular Languages

For example, the **natural numbers are closed under addition/multiplication** and **not under subtraction/division**.

Closure of Regular Languages

Q: Is the set of all regular languages **closed under union**?

Suppose L_1 and L_2 are regular languages. Is $L = L_1 \cup L_2$ also regular?



Set of all regular Languages

Closure of Regular Languages

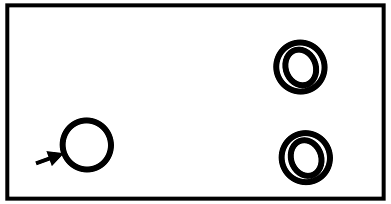
Q: Is the set of all regular languages **closed under union**?

Suppose L_1 and L_2 are regular languages. Is $L = L_1 \cup L_2$ also regular?

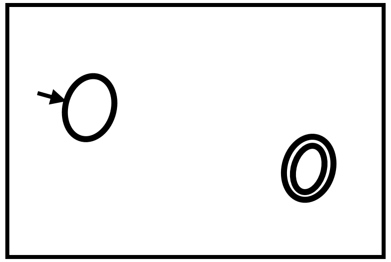
Proof: Since L_1 and L_2 are regular, there must be a DFA M_1 that accepts L_1, i.e. $L(M_1) = L_1$ and a DFA M_2 that accepts L_2, i.e. $L(M_2) = L_2$.

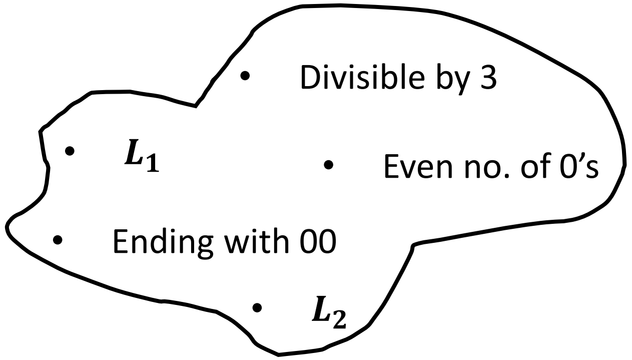
Using M_1 and M_2, we will show how to construct an NFA M that accepts $L = L_1 \cup L_2$, i.e. $L(M) = L_1 \cup L_2$.

Suppose the DFA for M_1 is



And the DFA for M_2 is





Set of all regular Languages

Closure of Regular Languages

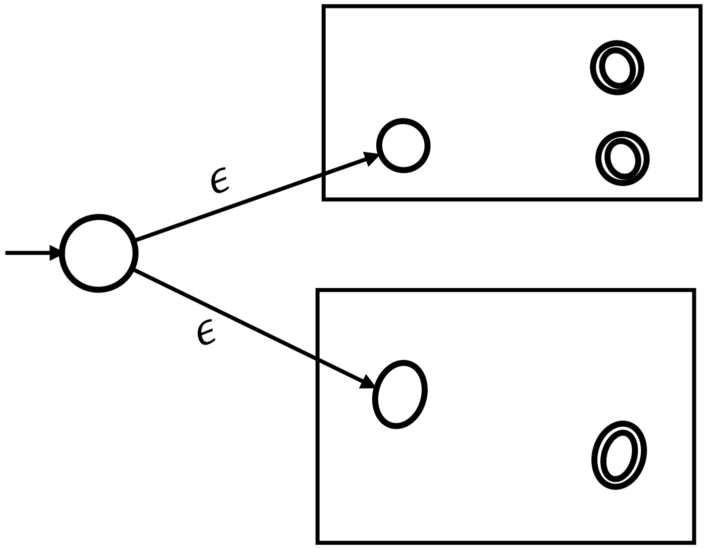
Q: Is the set of all regular languages **closed under union**?

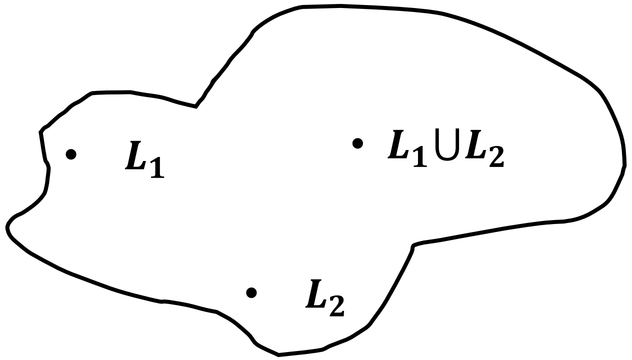
Suppose L_1 and L_2 are regular languages. Is $L = L_1 \cup L_2$ also regular?

Proof: Since L_1 and L_2 are regular, there must be a DFA M_1 that accepts L_1, i.e. $L(M_1) = L_1$ and a DFA M_2 that accepts L_2, i.e. $L(M_2) = L_2$.

Using M_1 and M_2, we will show how to construct an NFA M that accepts $L = L_1 \cup L_2$, i.e. $L(M) = L_1 \cup L_2$.

NFA M accepting $\boldsymbol{L = L_1 \cup L_2}$





Set of all regular Languages

Closure of Regular Languages

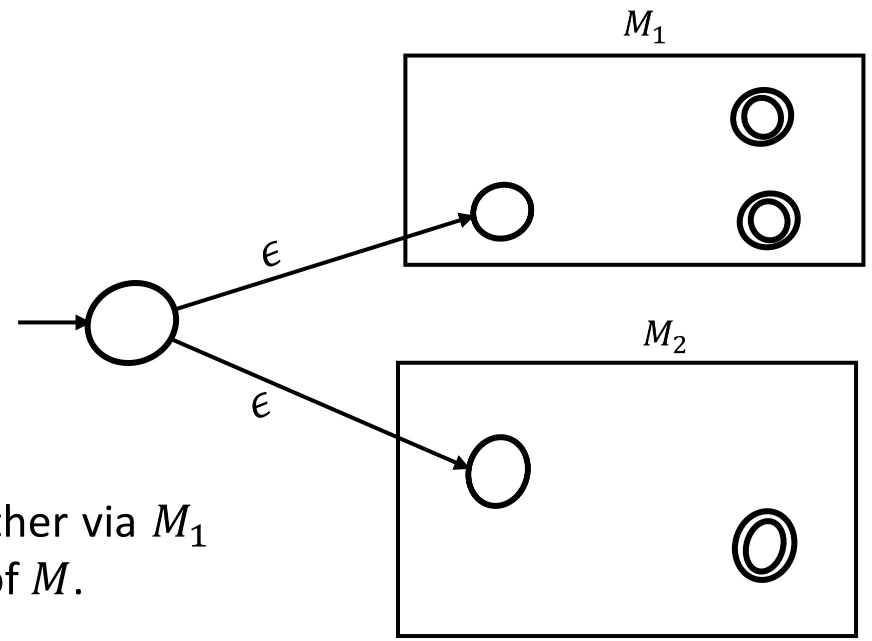
Q: Is the set of all regular languages **closed under union**?

Suppose L_1 and L_2 are regular languages. Is $L = L_1 \cup L_2$ also regular?

Proof: In order to prove that $L(M) = L_1 \cup L_2$, we show two things:

(i) $L \subseteq L_1 \cup L_2$

Let $\omega \in L$, i.e. ω is accepted by M. The final state for L can be reached either via M_1 or M_2. Thus ω must be accepted by either of them to reach the final state of M.



Closure of Regular Languages

Q: Is the set of all regular languages **closed under union**?

Suppose L_1 and L_2 are regular languages. Is $L = L_1 \cup L_2$ also regular?

Proof: In order to prove that $L(M) = L_1 \cup L_2$, we show two things:

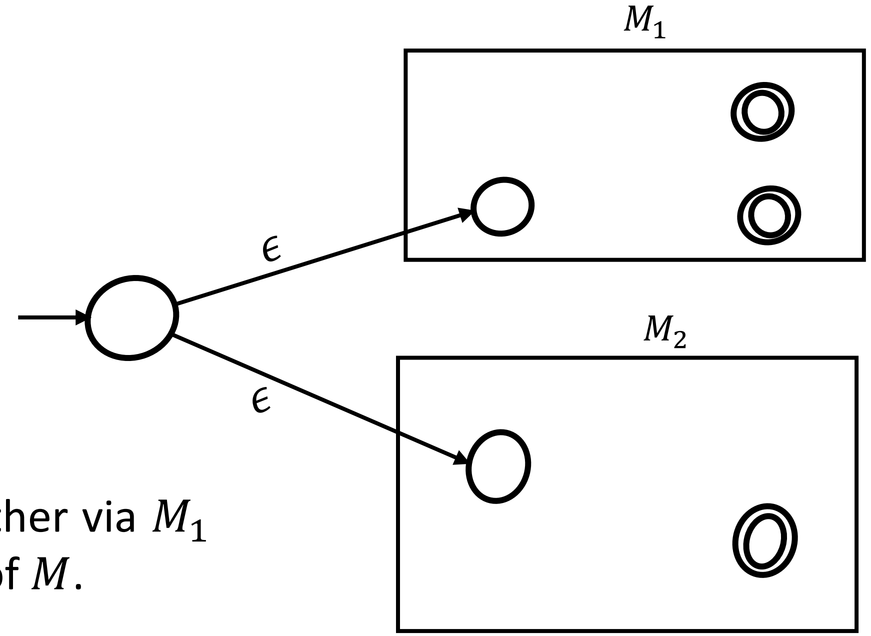
(i) $L \subseteq L_1 \cup L_2$

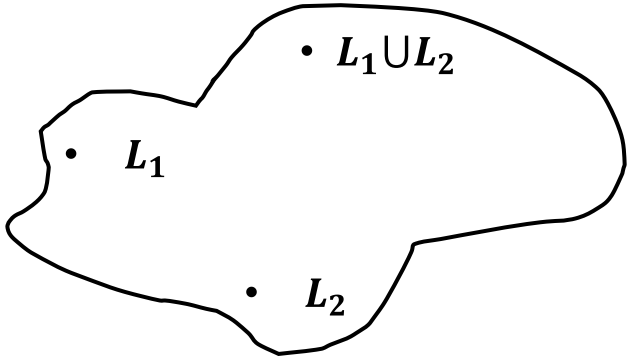
Let $\omega \in L$, i.e. ω is accepted by M. The final state for L can be reached either via M_1 or M_2. Thus ω must be accepted by either of them to reach the final state of M.

(ii) $L_1 \cup L_2 \subseteq L$

Let $\omega \in L_1 \cup L_2$. Then, $\omega \in L_1$ or $\omega \in L_2$.

Thus, ω must reach the final state of M_1 or M_2. But since the start state of M_1 or M_2 can be reached from the start state of M by taking an ϵ-transition, $\omega \in L$.





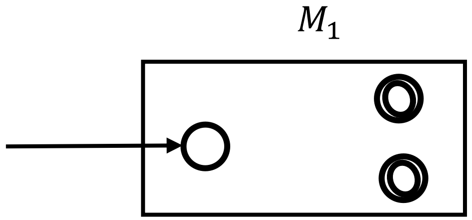
Set of all regular Languages

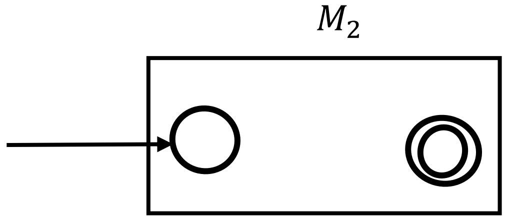
Closure of Regular Languages

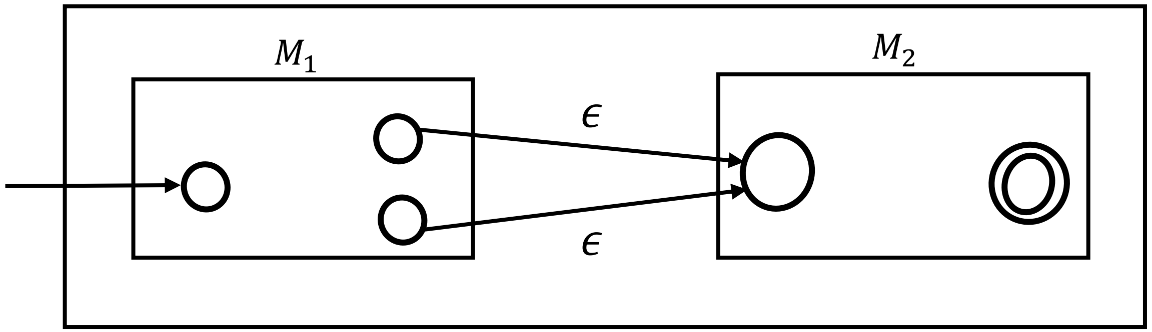
Q: Is the set of all regular languages **closed under concatenation**? Suppose L_1 and L_2 are regular languages. Is $L = L_1.L_2$ also regular?

Proof: Since L_1 and L_2 are regular, there must be a DFA M_1 that accepts L_1, i.e. $L(M_1) = L_1$ and a DFA M_2 that accepts L_2, i.e. $L(M_2) = L_2$**.**

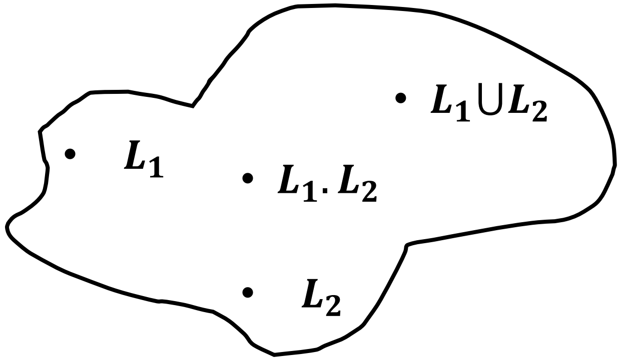
Using $M_1$and M_2, we will show how to construct an NFA M that accepts $L = L_1.L_2$.







NFA M accepting $\boldsymbol{L = L_1.L_2}$



Set of all regular Languages

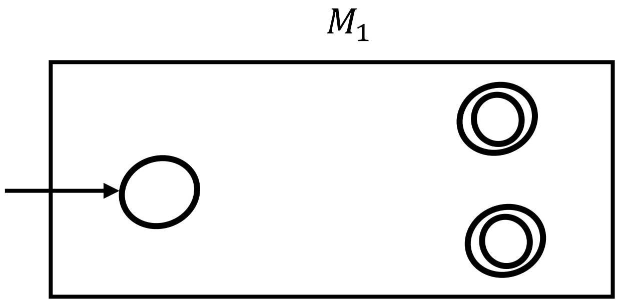
$L_1.L_2 = \{xy | x \in L_1 \text{ and } y \in L_2\}$

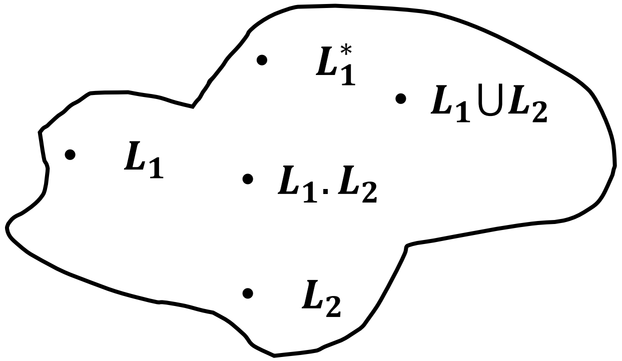
Closure of Regular Languages

Q: Is the set of all regular languages **closed under star**? Suppose L_1 is a regular language. Is L_1^* also regular?

Proof: Since L_1 is regular, there must be a DFA M_1 that accepts L_1, i.e. $L(M_1) = L_1$**.** Using M_1, we will show how to construct an NFA M that accepts $L = L_1^*$.

$L_1^* = \{x_1 x_2 \cdots x_k \mid k \geq 0 \text{ and each } x_i \in L_1\}$





Set of all regular Languages

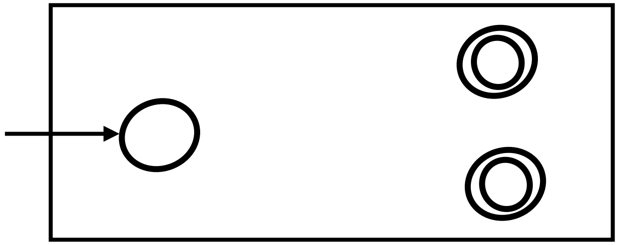
Closure of Regular Languages

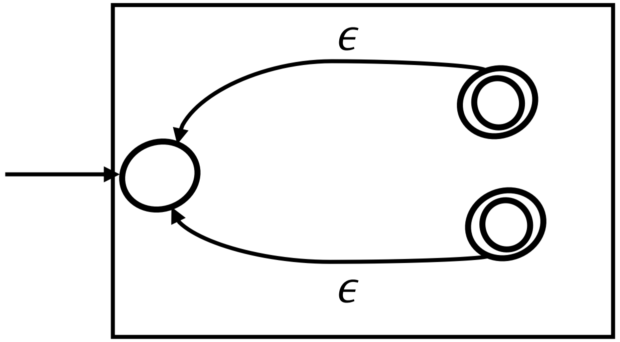
Q: Is the set of all regular languages **closed under star**? Suppose L_1 is a regular language. Is L_1^* also regular?

Proof: Since L_1 is regular, there must be a DFA M_1 that accepts L_1, i.e. $L(M_1) = L_1$**.** Using M_1, we will show how to construct an NFA M that accepts $L = L_1^*$.

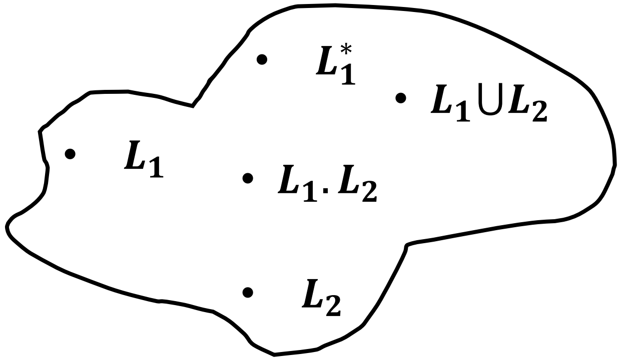
$L_1^* = \{x_1 x_2 \cdots x_k \mid k \geq 0 \text{ and each } x_i \in L_1\}$

M_1





NFA accepting $\boldsymbol{L = L_1^*}$



Set of all regular Languages

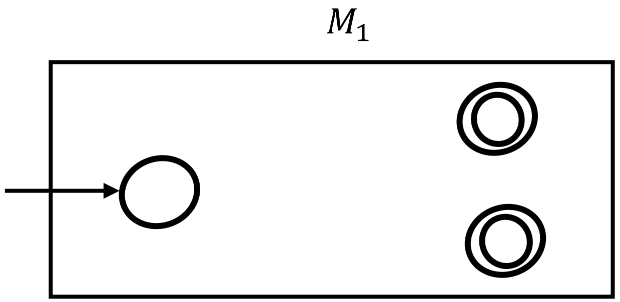
Steps:

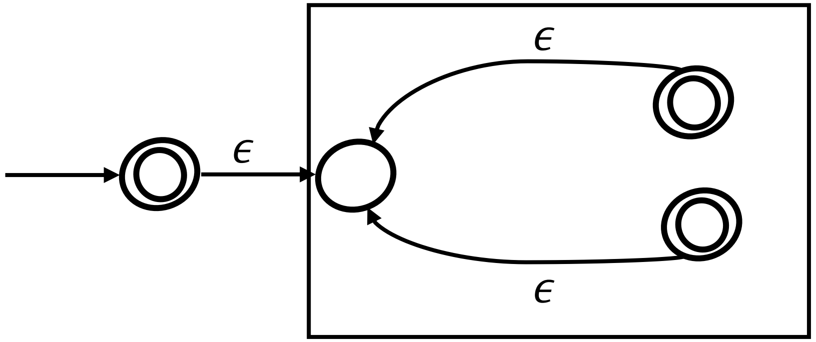
- Make ϵ-transitions from the final states of L_1 to the initial state of L_1.

Closure of Regular Languages

Q: Is the set of all regular languages **closed under star**? Suppose L_1 is a regular language. Is L_1^* also regular?

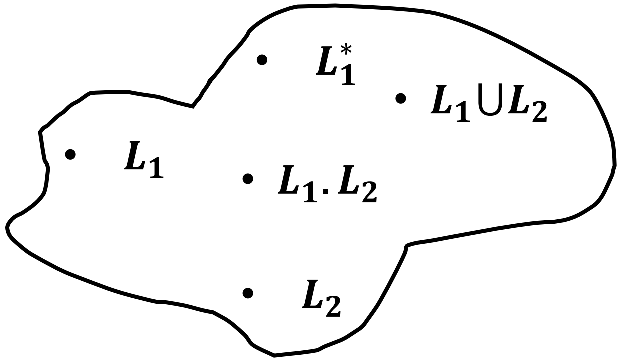
Proof: Since L_1 is regular, there must be a DFA M_1 that accepts L_1, i.e. $L(M_1) = L_1$**.** Using M_1, we will show how to construct an NFA M that accepts $L = L_1^*$.





NFA accepting $\boldsymbol{L = L_1^*}$

$L_1^* = \{x_1 x_2 \cdots x_k \mid k \geq 0$ and each $x_i \in L_1\}$



Set of all regular Languages

Steps:

- Make ϵ-transitions from the final states of L_1 to the initial state of L_1.
- Make a new final state as the start state and make an ϵ-transition from this state to the previous start state of L_1.

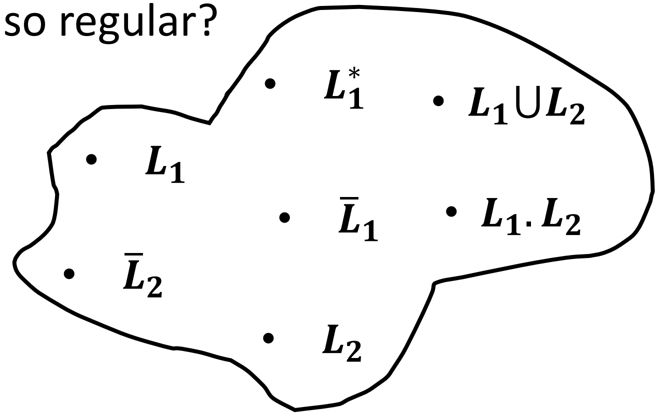
Closure of Regular Languages

Q: Is the set of all regular languages **closed under complement**? If L is regular, then is $\overline{L}$ also regular?

Proof: Given a DFA M, such that $L(M) = L$, construct the **toggled DFA** M' from M, by

(i) changing all the non-final states of M to be the final states of M' and
(ii) changing all the final states M to be the non-final states of M'.

$$\boldsymbol{L(M') = \overline{L}}$$



Set of all regular Languages

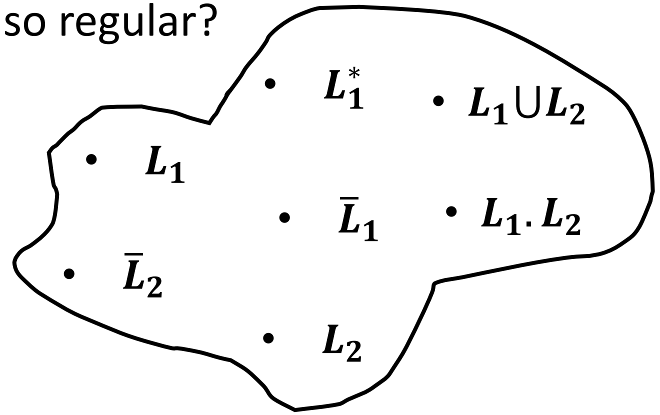
Closure of Regular Languages

Q: Is the set of all regular languages **closed under complement**? If L is regular, then is $\overline{L}$ also regular?

Proof: Given a DFA M, such that $L(M) = L$, construct the **toggled DFA** M' from M, by

(i) changing all the non-final states of M to be the final states of M' and
(ii) changing all the final states M to be the non-final states of M'.

$$\boldsymbol{L(M') = \overline{L}}$$

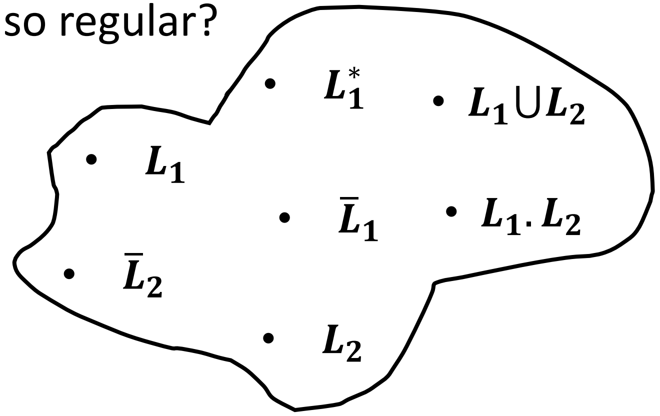


Set of all regular Languages

Q: If L is the language accepted by an NFA, does "toggling" its states result in an NFA that accepts $\overline{L}$?

Closure of Regular Languages

Q: Is the set of all regular languages **closed under complement**? If L is regular, then is $\overline{L}$ also regular?



Set of all regular Languages

Proof: Given a DFA M, such that $L(M) = L$, construct the **toggled DFA** M' from M, by

(i) changing all the non-final states of M to be the final states of M' and
(ii) changing all the final states M to be the non-final states of M'.

$$\boldsymbol{L(M') = \overline{L}}$$

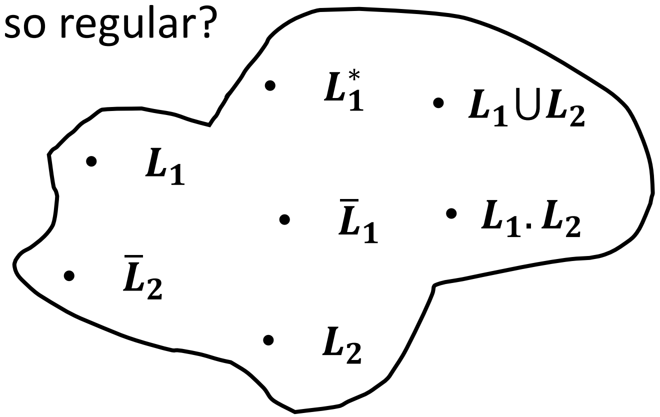
Q: If L is the language accepted by an NFA, does "toggling" its states result in an NFA that accepts $\overline{L}$?

Proof: Consider that for an input string $x \in L$**,** such that N accepts it. Suppose there is an rejecting run and an accepting run for input x. (See Table)

	NFA N	**Toggled NFA N'**
Run 1	Rejecting	
Run 2	Accepting	

Closure of Regular Languages

Q: Is the set of all regular languages **closed under complement**? If L is regular, then is $\overline{L}$ also regular?



Set of all regular Languages

Proof: Given a DFA M, such that $L(M) = L$, construct the **toggled DFA** M' from M, by

(i) changing all the non-final states of M to be the final states of M' and
(ii) changing all the final states M to be the non-final states of M'.

$$\boldsymbol{L(M') = \overline{L}}$$

Q: If L is the language accepted by an NFA, does "toggling" its states result in an NFA that accepts $\overline{L}$?

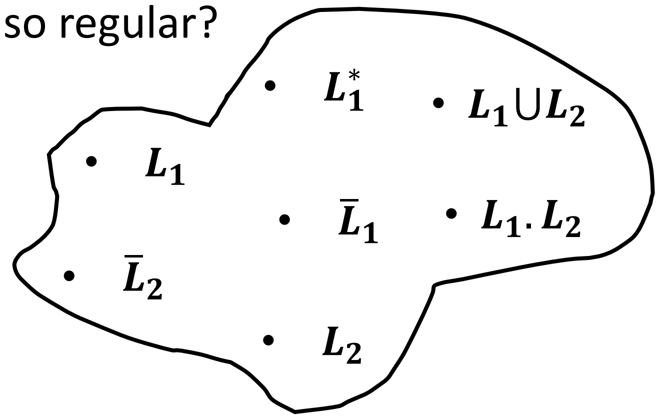
Proof: Consider that for an input string $x \in L$, such that N accepts it. Suppose there is an rejecting run and an accepting run for input x. (See Table)

For toggled NFA N' too, there are two runs for x. However, the rejecting run N is an accepting run for N'. Thus x is accepted by both N and N'.

	NFA N	**Toggled NFA N'**
Run 1	Rejecting	Accepting
Run 2	Accepting	Rejecting

Closure of Regular Languages

Q: Is the set of all regular languages **closed under complement**? If L is regular, then is $\overline{L}$ also regular?



Set of all regular Languages

Proof: Given a DFA M, such that $L(M) = L$, construct the **toggled DFA** M' from M, by

(i) changing all the non-final states of M to be the final states of M' and
(ii) changing all the final states M to be the non-final states of M'.

$$\boldsymbol{L(M') = \overline{L}}$$

Q: If L is the language accepted by an NFA, does "toggling" its states result in an NFA that accepts $\overline{L}$?

Proof: Consider that for an input string $x \in L$**,** such that N accepts it. Suppose there is an rejecting run and an accepting run for input x. (See Table)

For toggled NFA N' too, there are two runs for x. However, the rejecting run N is an accepting run for N'. Thus x is accepted by both N and N'.

Contradiction! So No, the **toggled NFA does not accept $\overline{L}$**.

	NFA N	**Toggled NFA N'**
Run 1	Rejecting	Accepting
Run 2	Accepting	Rejecting

Thank You!