Theory of Computing

5. Regular Languages

Professor Imed Bouchrika
National Higher School of Artificial Intelligence
imed.bouchrika@ensia.edu.dz

Outline :

- Regular Languages:
  - Finite Automata : DFA/NFA
  - Regular Expressions
- Operations on Regular Languages
- Pumping Lemma
- Distinguishable Strings and Fooling Sets

Regular Languages

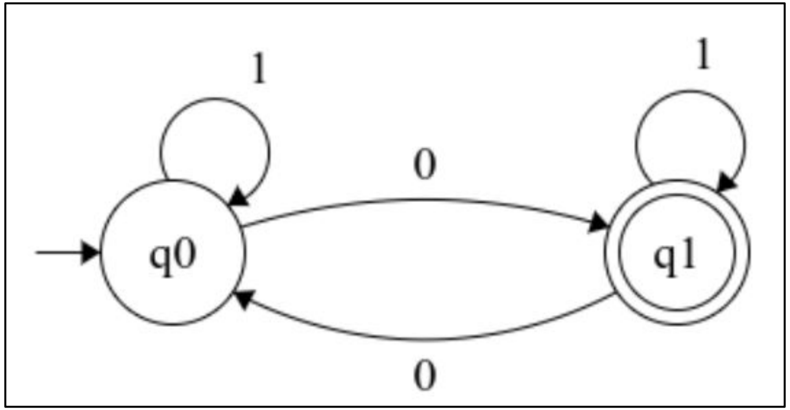
- **Deterministic Finite Automata : DFA**
  - A language which is represented by a DFA is a regular language

Regular Languages

- **Deterministic Finite Automata : DFA**
  - A language which is represented by a DFA is a regular language
  - Example :
    - L = { w | w contains an odd number of 0s } , Σ = { 0 ,1 }
    - Is a regular ?

Regular Languages

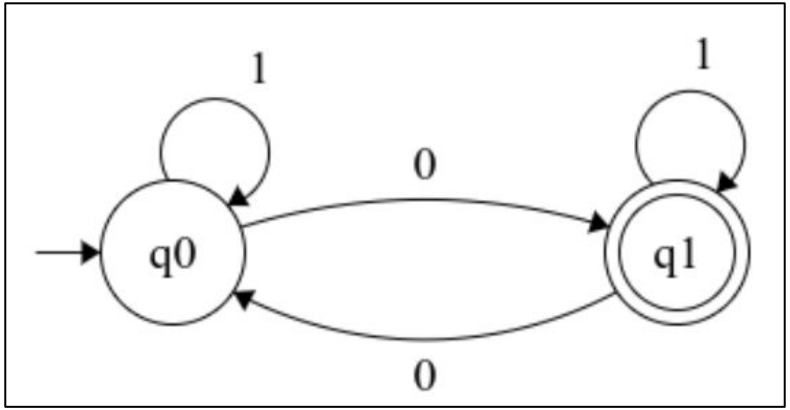
- **Deterministic Finite Automata : DFA**
  - A language which is represented by a DFA is a regular language
  - Example :
    - L = { w | w contains an odd number of 0s } , Σ = { 0   ,1 }
    - **Is a regular ?**



Regular Languages

- **Deterministic Finite Automata : DFA**
  - A language which is represented by a DFA is a regular language
  - Example :
    - L = { w | w contains an odd number of 0s } , Σ = { 0 ,1 }
    - **Is a regular ?**

It is a regular language because there is a DFA to represent the language



Regular Languages

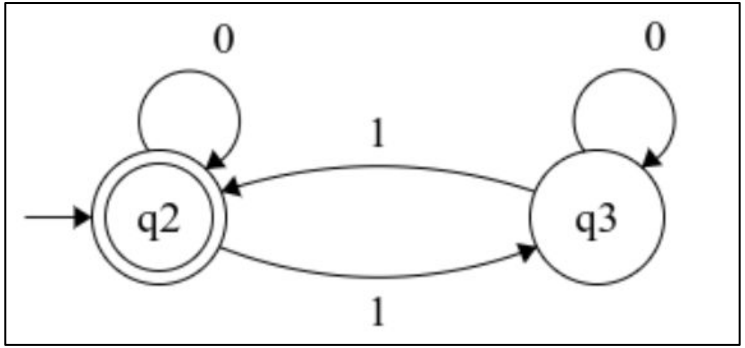
- **Nondeterministic Finite Automata : NFA**
    - A language which can be represented by a nondeterministic finite automaton is a regular language

Regular Languages

- **Nondeterministic Finite Automata : NFA**
  - A language which can be represented by a nondeterministic finite automaton is a regular language
  - Example:
    - L = { w | w contains an even number of 1s } , Σ = { 0 ,1 }
    - Is a regular ?

Regular Languages

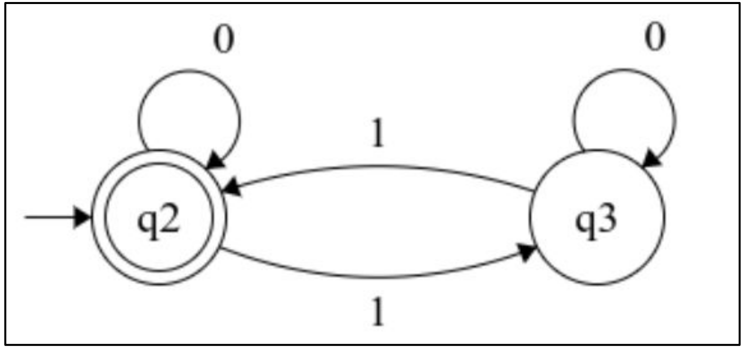
- **Nondeterministic Finite Automata : NFA**
  - A language which can be represented by a nondeterministic finite automaton is a regular language
  - Example:
    - L = { w | w contains an even number of 1s } , Σ = { 0 ,1 }
    - **Is a regular ?**



Regular Languages

- **Nondeterministic Finite Automata : NFA**
  - A language which can be represented by a nondeterministic finite automaton is a regular language
  - Example:
    - L = { w | w contains an even number of 1s } , Σ = { 0 ,1 }
    - **Is a regular ?**

It is a regular language because there is NFA to represent the language



Regular Languages

- **Regular Expression**
  - A language which can be represented by a regular expression is a regular language. Remember the formal definition.

Regular Languages

- **Regular Expression**
  - A language which can be represented by a regular expression is a regular language.
  - Example :
    - L = { w | w contains exactly two 0's.} , Σ = { 0 ,1 }

Regular Languages

- **Regular Expression**
  - A language which can be represented by a regular expression is a regular language.
  - Example :
    - L = { w | w contains exactly two 0's.} , Σ = { 0 ,1 }
    - $1^*01^*01^*$

Regular Languages

- **Regular Expression**
  - A language which can be represented by a regular expression is a regular language.
  - Example :
    - L = { w | w contains 11 as a substring.} , Σ = { 0 ,1 }

Regular Languages

- **Regular Expression**
  - A language which can be represented by a regular expression is a regular language.
  - Example :
    - L = { w | w contains 11 as a substring.} , Σ = { 0 ,1 }
    - $\{0,1\}^*11\{0,1\}^*$
    - Is it regular ? It is.

Regular Languages

- **Regular Expression**
  - A language which can be represented by a regular expression is a regular language.
  - Example :
    - L = { w | w **does not** contain 11 as a substring.} , Σ = { 0 ,1 }

Regular Languages

- **Regular Expression**
  - A language which can be represented by a regular expression is a regular language.
  - Example :
    - L = { w | w **does not** contain 11 as a substring.} , Σ = { 0 ,1 }
    - $\{0,10\}^{*}\{\boldsymbol{\varepsilon},1\}$ == $(0 \mid 10)^{*} (\boldsymbol{\varepsilon} \mid 1)$
    - Is it regular ? **Yes , it is.**

Operations on Regular Languages

- **Closure Properties**
  - Let L_1 and L_2 be regular languages. Then, the following languages are regular.
    - **Complement :** $L'_1 = \{x \mid x \in \Sigma^* \text{ and } x \notin L_1\}$.
    - **Union :** $L_1 \cup L_2 = \{x \mid x \in L_1 \text{ or } x \in L_2\}$.
    - **Intersection :** $L_1 \cap L_2 = \{x \mid x \in L_1 \text{ and } x \in L_2\}$.
    - **Concatenation :** $L_1 \cdot L_2 = \{xy \mid x \in L_1 \text{ and } y \in L_2\}$.
    - **Star :** $L_1^* = \{x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in L_1\}$.

Language	Operation $L_1 \cup L_2$	$L_1 \cap L_2$	$\bar{L}$	$L_1 \circ L_2$	L^*
DFA	Easy	Easy	Easy	Hard	Hard
Regex	Easy	Hard	Hard	Easy	Easy
NFA	Easy	Hard	Hard	Easy	Easy

- $L_1 \cup L_2 =$ Union of L_1 and L_2
- $L_1 \cap L_2 =$ Intersection of L_1 and L_2
- $\bar{L} =$ Complement of L
- $L_1 \circ L_2 =$ Concatenation of L_1 and L_2
- $L^* =$ Powers of L

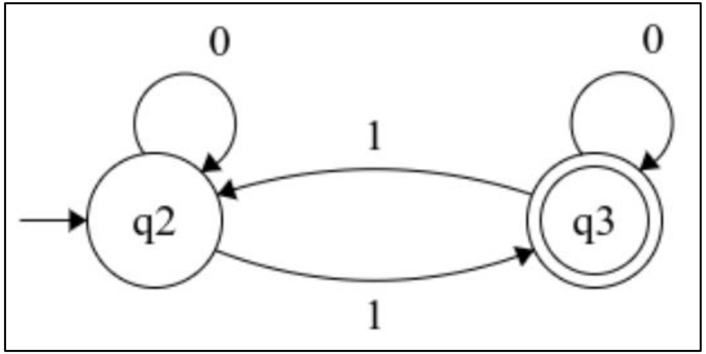
Taken from Pramod Ganapathi Department of Computer Science, State University of New York at Stony Brook

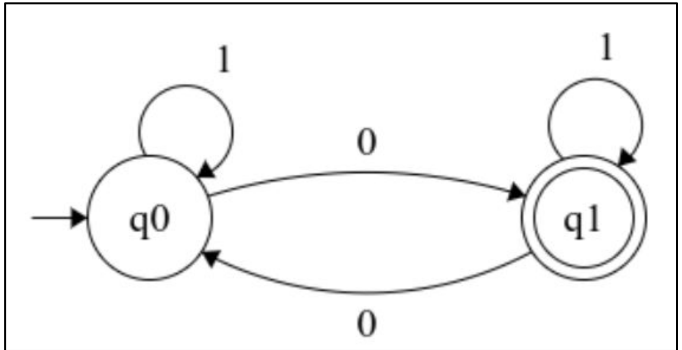
Operations on Regular Languages

- **Example**
  - $L_1 = \{ w \mid w$ contains an even number of 1s $\}$, $\Sigma = \{ 0, 1 \}$
  - $L_2 = \{ w \mid w$ contains an odd number of 0s $\}$, $\Sigma = \{ 0, 1 \}$

Operations on Regular Languages

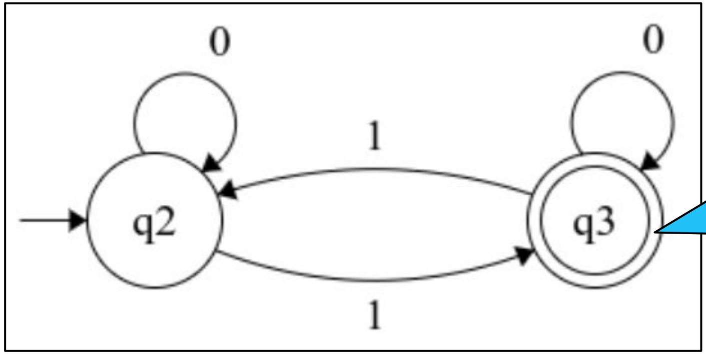
- **Union**
  - NFA :
    - *L = { w | w contains an even number of 1s **or** an odd number of 0s }*
      - *L_1 = { w | w contains an even number of 1s } , Σ = { 0 ,1 }*
      - *L_2 = { w | w contains an odd number of 0s } , Σ = { 0 ,1 }*



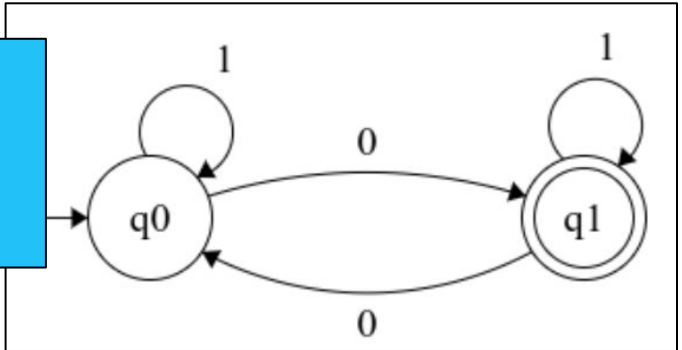


Operations on Regular Languages

- **Union**
  - NFA :
    - *L = { w | w contains an even number of 1s **or** an odd number of 0s }*
      - ***L_1 = { w | w contains an even number of 1s } , Σ = { 0 ,1 }***
      - *L_2 = { w | w contains an odd number of 0s } , Σ = { 0 ,1 }*

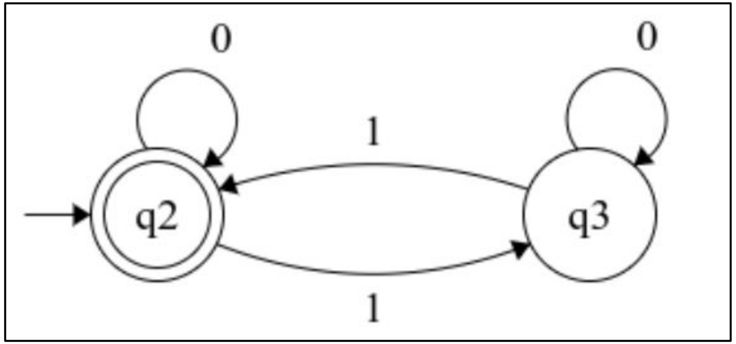


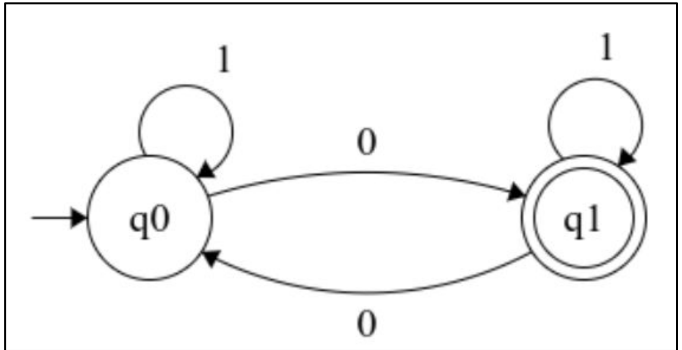
Incorrect



Operations on Regular Languages

- **Union**
  - NFA :
    - *L = { w | w contains an even number of 1s* ***or*** *an odd number of 0s }*
      - L_1 = *{ w | w contains an even number of 1s }* , $\Sigma = \{ 0 \;\; ,1 \}$
      - L_2 = *{ w | w contains an odd number of 0s }* , $\Sigma = \{ 0 \;\; ,1 \}$



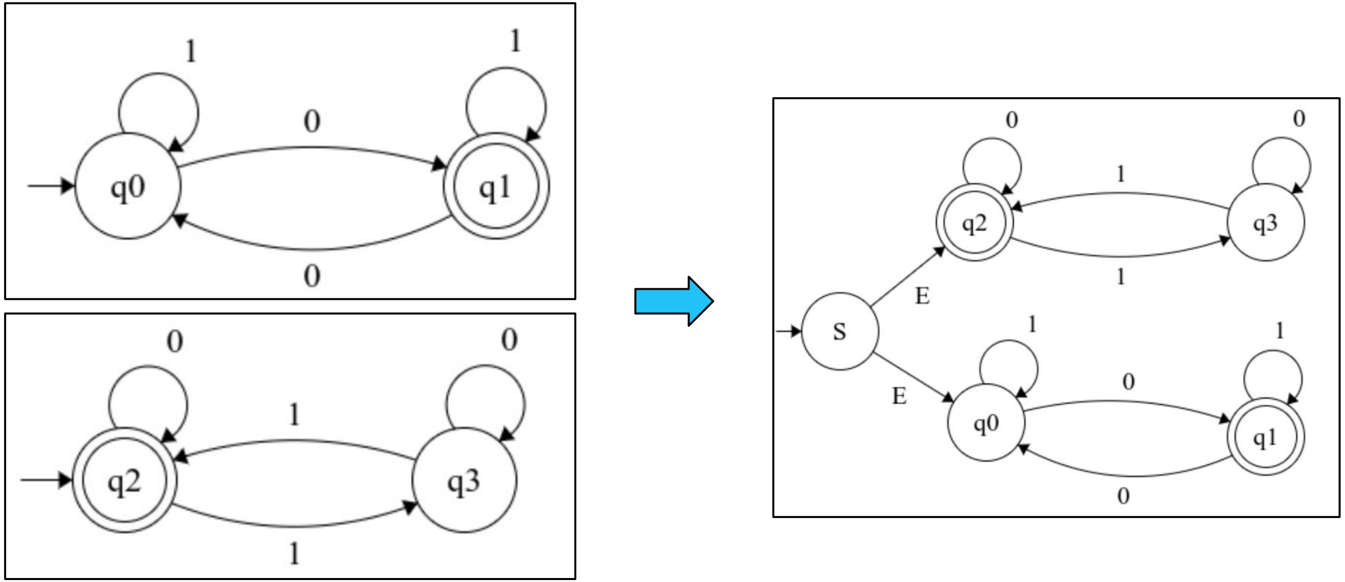


Operations on Regular Languages

- **Union**
  - NFA :
    - **Steps**
      - **Create a new initial start state**
      - **Link it to both start states with Epsilon**

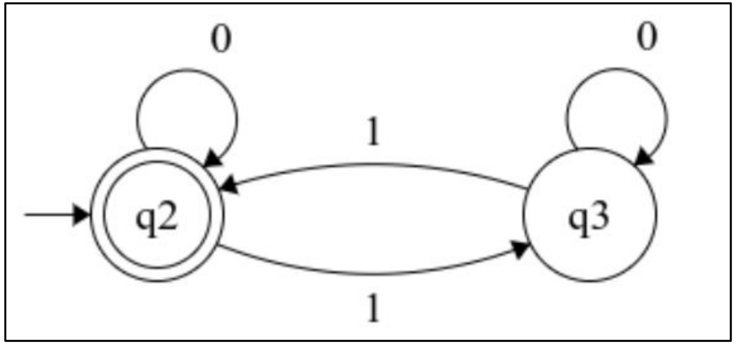
Operations on Regular Languages

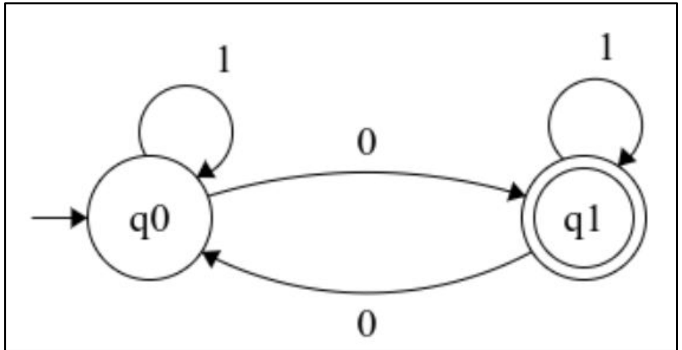
- **Union**
  - NFA :



Operations on Regular Languages

- **Union**
  - DFA :
    - *L = { w | w contains an even number of 1s* ***or*** *an odd number of 0s }*
      - L_1 = *{ w | w contains an even number of 1s }* , Σ = { 0 ,1 }
      - L_2 = *{ w | w contains an odd number of 0s }* , Σ = { 0 ,1 }



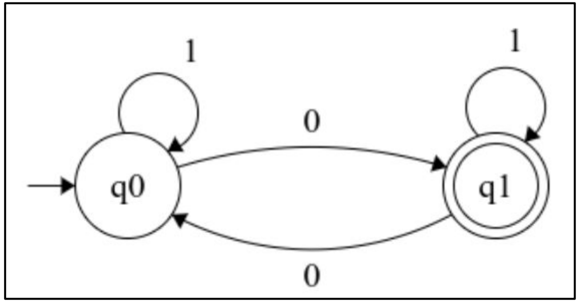


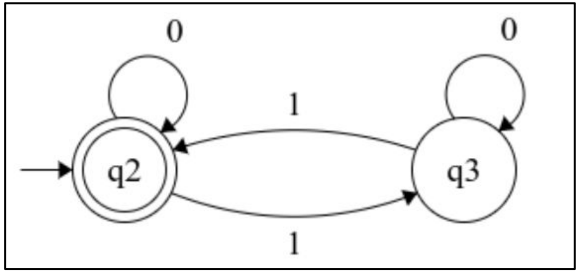
Operations on Regular Languages

- **Union**
  - DFA :
    - Steps:
      - *Create new states represented by Q_1 x Q_2*
      - *Accepting State : any pair/tuple containing an original accept state*
      - *Start State : is the pair/tuple containing both start states.*
      - *Transitions : for a pair of states (q0 , q1) upon reading a, see where to move for q0 and q1, the results would be the pair of states.*

Operations on Regular Languages

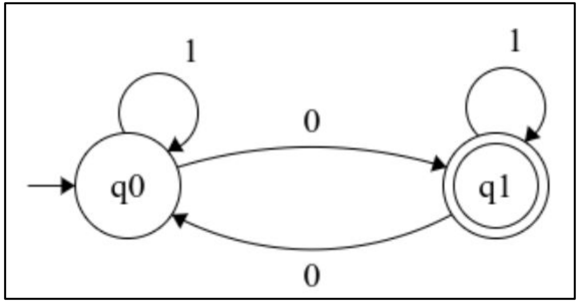
- **Union**
  - DFA :

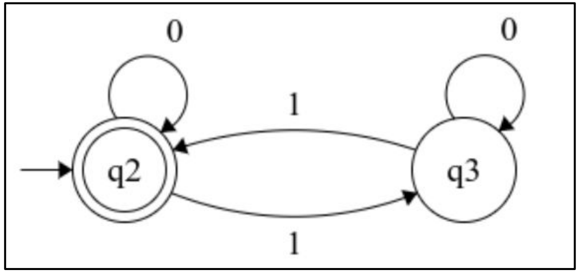


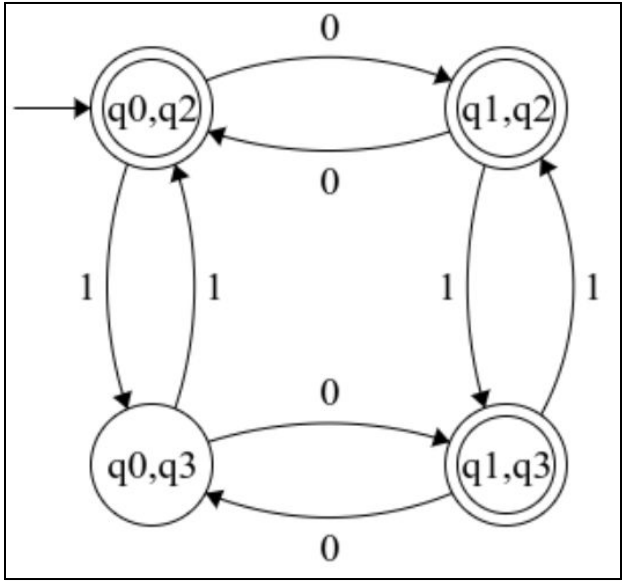


Operations on Regular Languages

- **Union**
  - DFA :







Operations on Regular Languages

- **Union**
  - Regular Expressions :
    - *L = { w | w contains an even number of 1s* ***or*** *an odd number of 0s }*
      - L_1 *= { w | w contains an even number of 1s } , Σ = { 0 ,1 }*
      - L_2 *= { w | w contains an odd number of 0s } , Σ = { 0 ,1 }*
    - *RE(*L_1*) =* `(0* 1 (0)* 1 (0)* )*`
    - *RE(*L_2*) =* `1* 0 (1)* ( 1* 0 1* 0 1* )*`

Operations on Regular Languages

- **Union**
  - Regular Expressions :
    - *L = { w | w contains an even number of 1s* **or** *an odd number of 0s }*
      - L_1 *= { w | w contains an even number of 1s } ,* Σ *= { 0 ,1 }*
      - L_2 *= { w | w contains an odd number of 0s } ,* Σ *= { 0 ,1 }*
    - $RE(L_1)$ = `( 0* 1 (0)* 1 (0)* )*`
    - $RE(L_2)$ = `1* 0 (1)* ( 1* 0 1* 0 1* )*`
      - $RE(L)$ = `( 1 (0)* 1 (0)* )* | 1* 0 (1)* ( 1* 0 1* 0 1* )*`

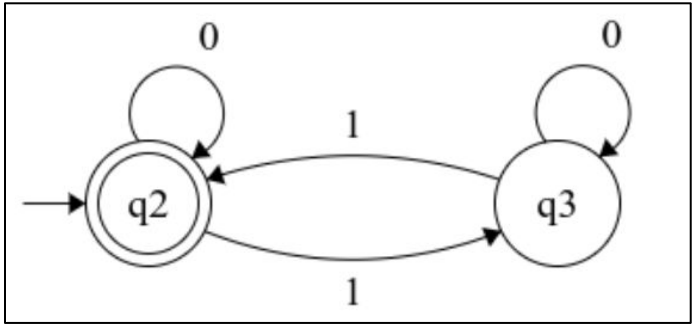
Language	Operation				
	$L_1 \cup L_2$	$L_1 \cap L_2$	$\bar{L}$	$L_1 \circ L_2$	L^*
DFA	Easy	Easy	Easy	Hard	Hard
Regex	Easy	Hard	Hard	Easy	Easy
NFA	Easy	Hard	Hard	Easy	Easy

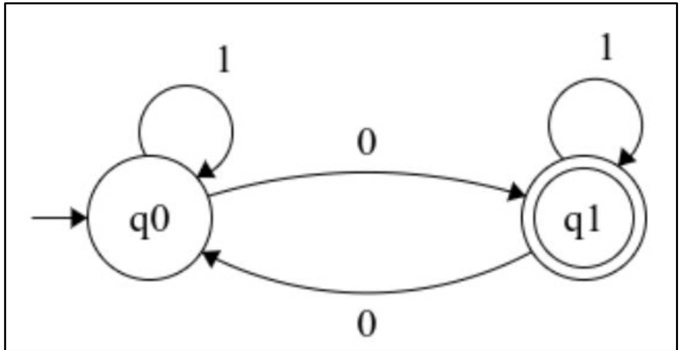
- $L_1 \cup L_2 =$ Union of L_1 and L_2
- $L_1 \cap L_2 =$ Intersection of L_1 and L_2
- $\bar{L} =$ Complement of L
- $L_1 \circ L_2 =$ Concatenation of L_1 and L_2
- $L^* =$ Powers of L

Taken from Pramod Ganapathi Department of Computer Science, State University of New York at Stony Brook

Operations on Regular Languages

- **Intersection**
  - DFA :
    - *L = { w | w contains an even number of 1s **and** an odd number of 0s }*
      - *L_1 = { w | w contains an even number of 1s } , Σ = { 0 ,1 }*
      - *L_2 = { w | w contains an odd number of 0s } , Σ = { 0 ,1 }*



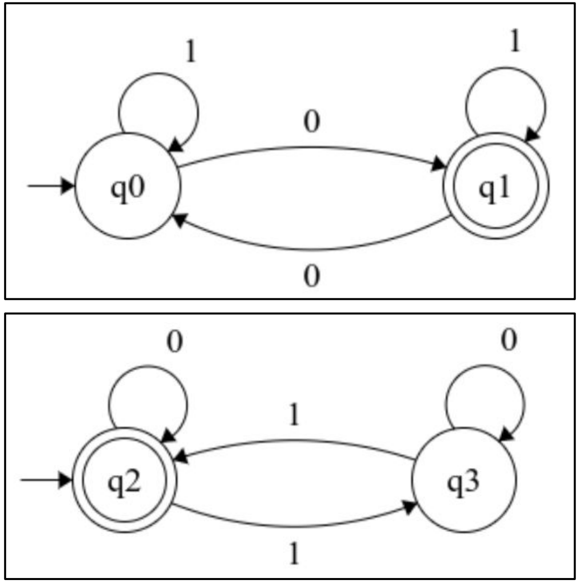


Operations on Regular Languages

- **Intersection**
  - DFA :
    - Steps:
      - *Create new states represented by $Q_1 \times Q_2$*
      - *Accepting State : pairs/tuples containing **both** original accept states*
      - *Start State : is the pair/tuple containing both start states.*
      - *Transitions : for a pair of states (q0 , q1) upon reading a, see where to move for q0 and q1, the results would be the pair of states.*

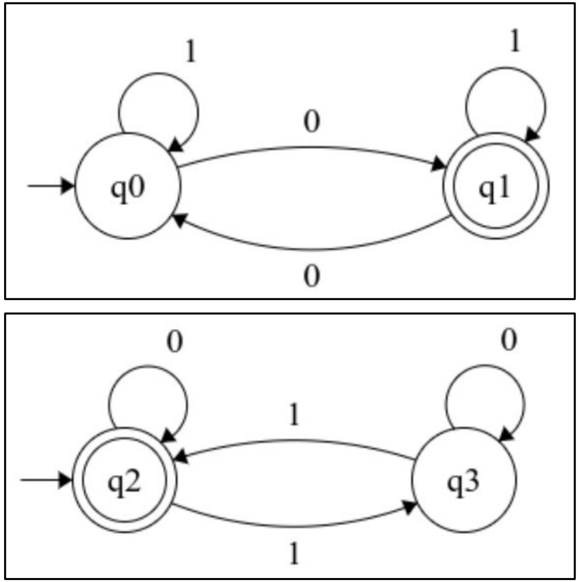
Operations on Regular Languages

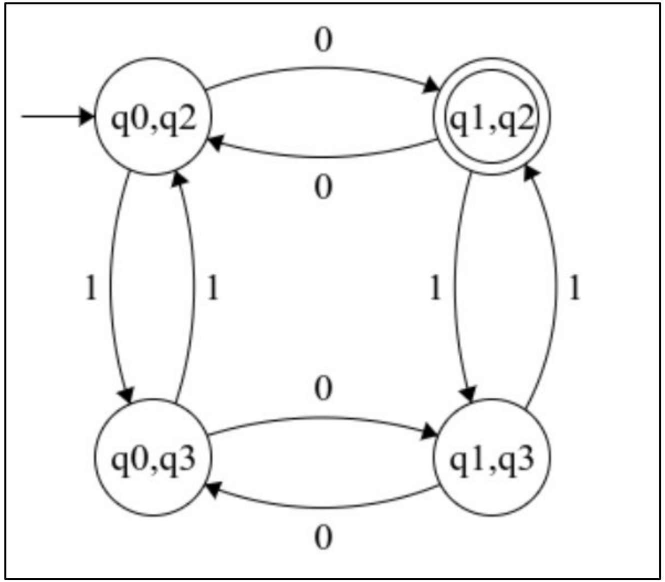
- **Intersection**
  - DFA :



Operations on Regular Languages

- **Intersection**
    - DFA :





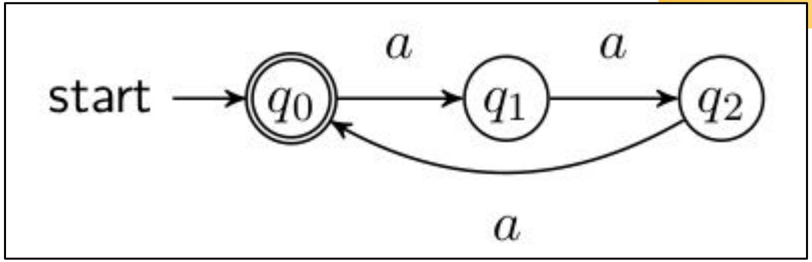
Operations on Regular Languages

- **Intersection**
  - NFA
    - You can convert the NFA to DFA and do the intersection
    - Or:
      - Do it the same way in addition to adding also ε-transition when relevant
  - Regular Expressions:
    - Very hard to do it directly. A naive approach is to convert to DFAs…

Operations on Regular Languages

- **Complement**
  - DFA
    - Inverse Accepting to Non-Accepting states and vice versa
    - Example:
      - M is the automaton for the language L = { w | the length of w is divisible by 3 }
      - Alphabet is { a }
      - Language L: {, aaa, aaaaaa, aaaaaaaaa, . . .}

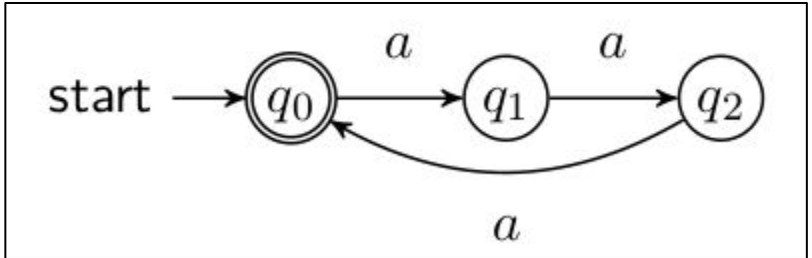
Operations on Regular Languages



- **Complement**
  - DFA
    - Inverse Accepting to Non-Accepting states and vice versa
    - Example:
      - M is the automaton for the language L = { w | the length of w is divisible by 3 }
      - Alphabet is { a }
      - Language L: {, aaa, aaaaaa, aaaaaaaaa, . . .}

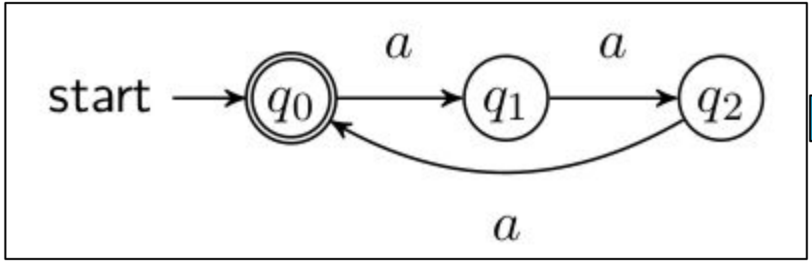
Operations on Regular Languages

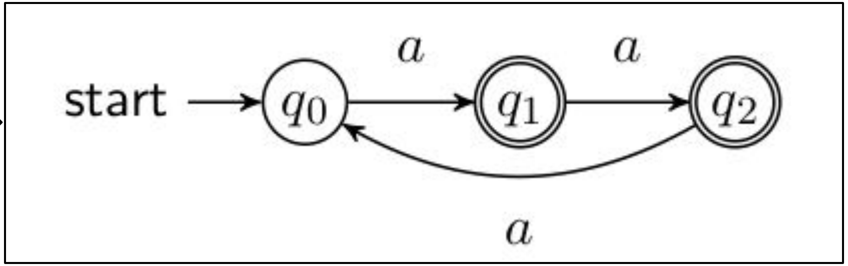
- **Complement**
  - DFA
    - What's the automaton for the **complement**
      - L' = { w | the length of w is **not** divisible by 3 }
      - Alphabet is { a }
      - Language L = { a, aa, aaaaaa, aaaaaaaaa, . . .}



Operations on Regular Languages

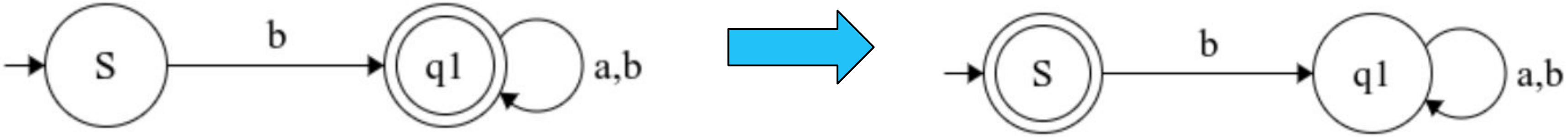
- **Complement**
  - DFA
    - What's the automaton for the **Complement**
      - L' = { w | the length of w is **not** divisible by 3 }
      - Alphabet is { a }
      - Language L = { a, aa, aaaaaa, aaaaaaaaa, . . .}





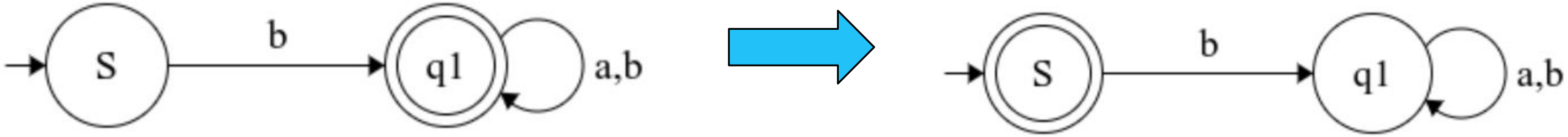
Operations on Regular Languages

- **Complement**
  - NFA
    - The method described for DFA does not always work
    - For example : language represented by b(a|b)*



Operations on Regular Languages

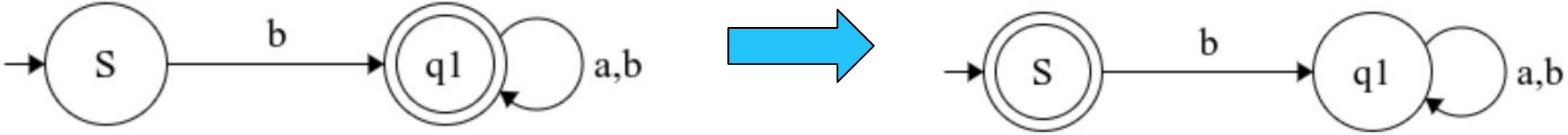
- **Complement**
  - NFA
    - The method described for DFA does not always work
    - For example : language represented by b(a|b)*
      - The complement for the NFA is not correct because ?



Operations on Regular Languages

- **Complement**
  - NFA
    - The method described for DFA does not always work
    - For exar
      - The ... because ?

Missing Transitions/Trap states are not considered for NFA.



Operations on Regular Languages

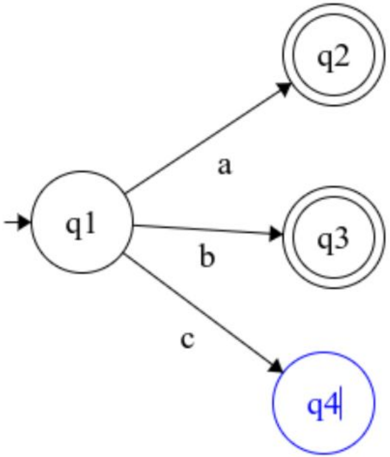
- **Complement**
  - Regular Expressions:
    - You have to design it from scratch. (Of course, there is the **not** operator in the regular expressions being used for text processing)

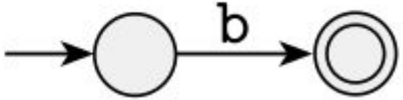
Operations on Regular Languages

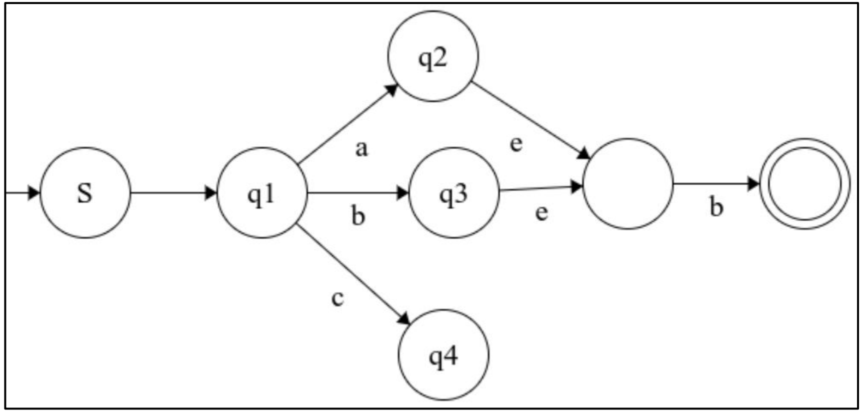
- **Concatenation**
  - NFA
    - Seen in the previous lecture
      - ***Link Accepting States of A to Start state of B with epsilon transition***
      - ***Convert all Accepting states of A to non-accepting***

Operations on Regular Languages

- **Concatenation**
  - NFA

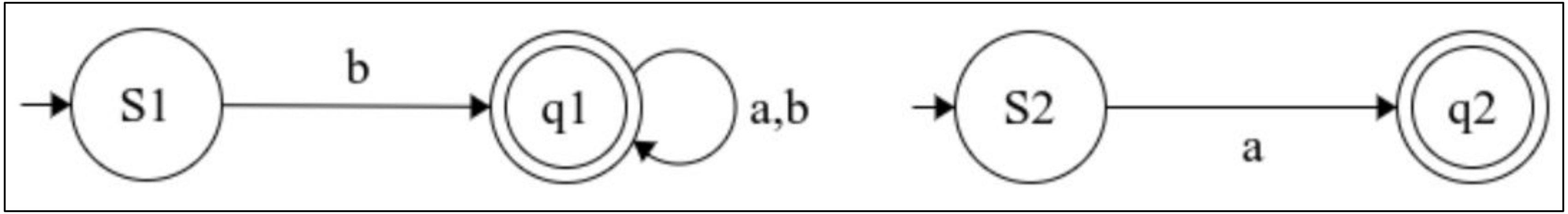






Operations on Regular Languages

- **Concatenation**
    - Regular Expressions :
        - Easy, as it is part of it.
    - DFA :
        - Extremely difficult, need to do it as NFA instead.
            - How to concatenate the following two DFAs



Non-Regular Languages

- **Questions**
  - Are all languages regular ?
  - Can we create DFA/NFA/Regular Expression for any Language
  - Remember :
    - ***Finite state machines have a limited amount of memory***
    - ***Why it is called : finite state ?***

Non-Regular Languages

- **Questions**
  - Are all languages regular ?
    - ***There are other languages that we call them non-regular languages***
  - Can we create DFA/NFA/Regular Expression for any Language
    - ***No, there are other languages that may require more memory.***
  - Finite States :
    - ***Remember : DFA or NFA cannot have infinite number of states***

Non-Regular Languages

- **Questions**
  - Is the language for all English **union** French words regular over the latin alphabet ?
  - Is the language containing odd 1s and even 0s regular over alphabet {0,1}?
  - Is the language in the form : ***wordword*** regular over any alphabet ?
  - Is the language in the form : ***wordword*** regular over { 0, 1 } such that |word|=1
  - Is the language of alternating 0 and 1 in a word regular (01, 010,1010,...) ?

Non-Regular Languages

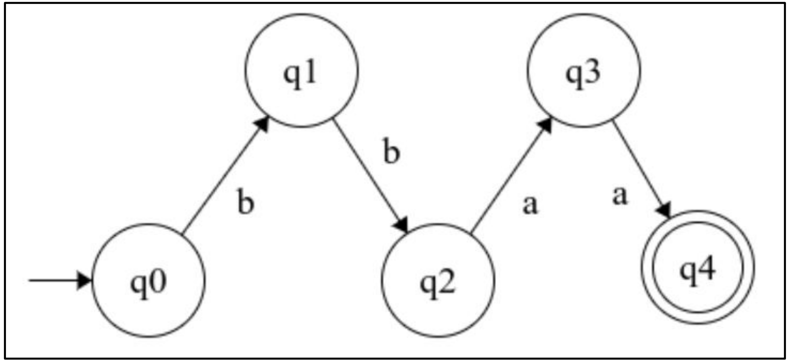
- **Questions**
  - Is the language for all English **union** French words regular over the latin alphabet ?
    - Yes, Because we can build NFA for each word -> do the union for all words.
  - Is the language containing odd 1s and even 0s regular over alphabet {0,1}?
    - Yes, we have designed the DFA for it.

Non-Regular Languages

- **Questions**
  - Is the language of alternating 0 and 1 in a word regular (01, 010,1010,...) ?
    - It is , because we can have the regular expression:
      - $(01)^* \mid (10)^*$
  - Is the language in the form : ***wordword*** regular over any alphabet ?
    - No, because we need to **remember the sequence of symbols for the first word (need extra memory)** so that we repeat it in the next word.

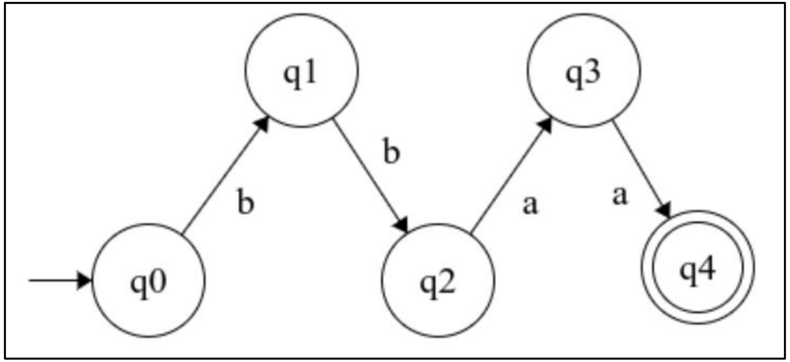
Non-Regular Languages
Pumping Lemma

- **Pumping Words**
  - Given a finite state machine of N states (suppose N=5):
    - Finite number of states
    - **2 Questions:**
      - Max length of strings ?
      - Max number of strings ?

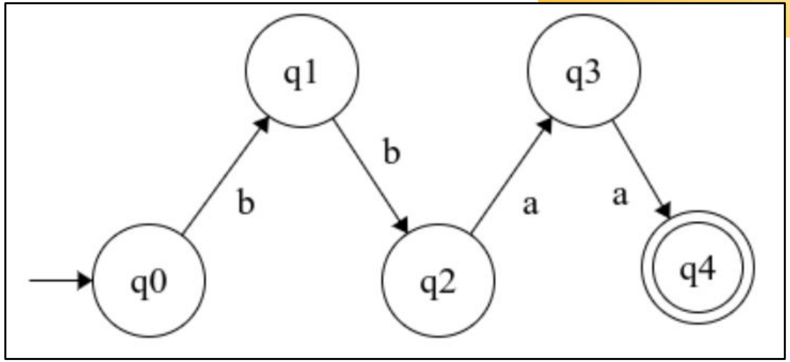


Non-Regular Languages
Pumping Lemma

- **Pumping Words**
  - Given a finite state machine of N states (suppose N=5):
    - Finite number of states
    - **2 Questions:**
      - Max length of strings ? **4**
      - Max number of strings ? **1**



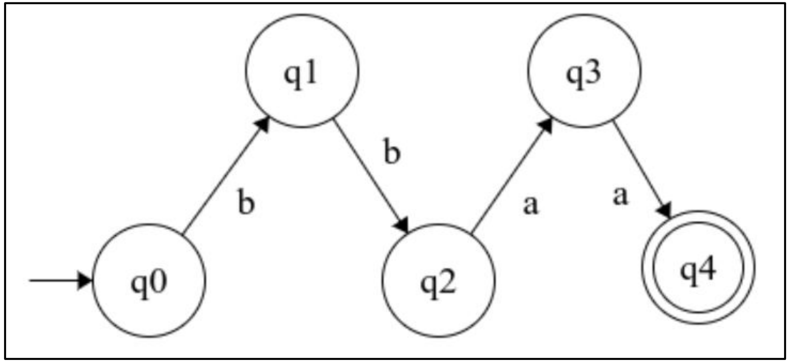
Non-Regular Languages
Pumping Lemma

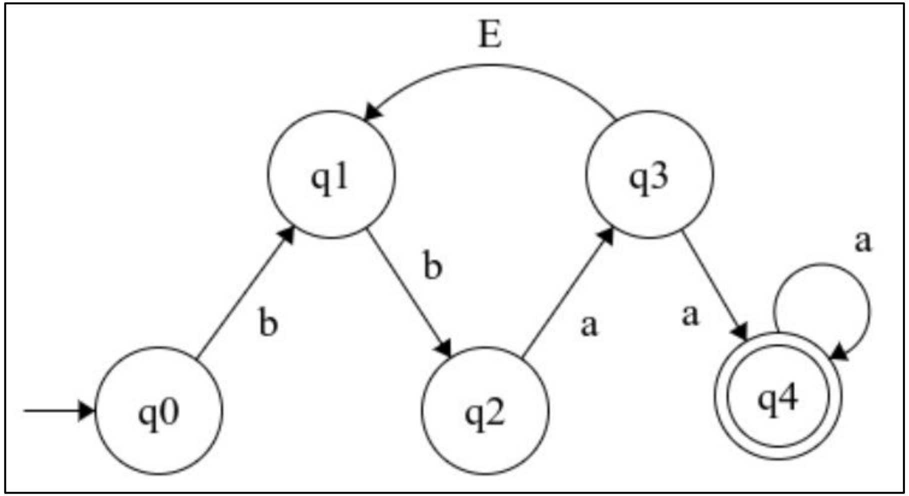


- **Pumping Words**
  - Given a finite state machine of N states
    - Finite number of states
    - **2 Questions:**
      - **When a state machine accepts Strings with length > The number of states ?**
      - **Number of states is finite : How to create a language with infinite words ?**

Non-Regular Languages
Pumping Lemma

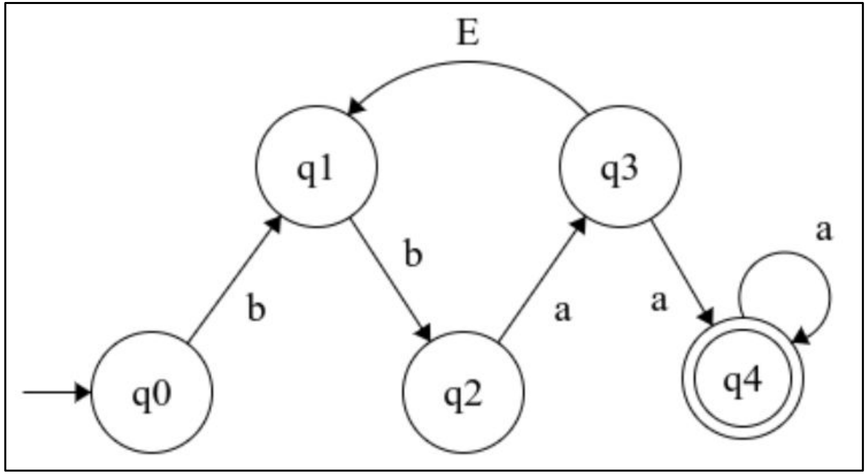
- **Pumping Words**
  - Given a finite state machine of N states (suppose N=5):





Non-Regular Languages
Pumping Lemma

- **Pumping Words**
  - Given a finite state machine of N states (suppose N=5):
    - We can generate:
      - bbaa
      - bba**ba**aa
      - bba**baba**aa
      - bba**bababa**aa
      - Bba**babababa**aa
      - bba**bababababa........**aa**a...**
    - Infinite number + infinite length



Non-Regular Languages
Pumping Lemma

- **Pumping Words**
  - Given a finite state machin
    - We can generate:
      - bbaa
      - bba**ba**aa
      - bba**baba**aa
      - bba**bababa**aa
      - Bba**babababa**aa
      - bba**babababa**
    - Infinite number + infinite length

It means :

For any language, there should be some number N (Assume the number of states)

If a string with length > N,

There must be some pumping for a given symbol or substring ? so that we have a bigger string ?

Non-Regular Languages
Pumping Lemma

- **Pumping Words**
  - Given a finite state machin
    - We can generate:
      - bbaa
      - bba**ba**aa
      - bba**baba**aa
      - bba**bababa**aa
      - bba**babababa**aa
      - bba**bababababa........**aa**a...**
    - Infinite number + infinite length

It means formally:

There is string s: can be written into three parts :

s= xyz

which is in L

At the same time :

xy^2z , xy^3z , xy^4z ,.... are in the language L

Languages
Lemma

y =ba
y^2 =baba
y^3 =bababa
...

- Given a finite state machin
  - We can generate:
    - bbaa
    - bba**ba**aa
    - bba**baba**aa
    - bba**bababa**aa
    - bba**babababa**aa
    - bba**babababa........**aa**a...**
  - Infinite number + infinite length

It means formally:

The big string s: can be split into three parts :

s= xyz

which is in L

At the same time :

xy^2z , xy^3z , xy^4z ,... are in the language L

Question :

What if you have an infinite language L

You are given the string (Example bbabaaa) in L

But

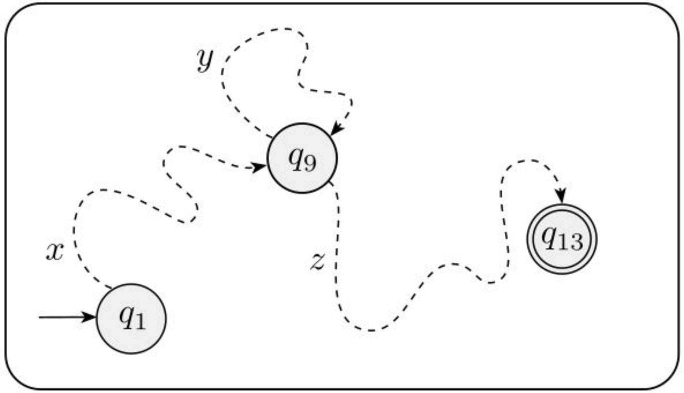
You cannot find some part (let's call it Y) from that string so that regardless of how you pump Y (repeat), the newly generated string is not in the language ?

**The example need to be taken consider traversing a loop with a single iteration*

- bba**babababababa........**aa**a...**
- Infinite number + infinite length

Non-Regular Languages
Pumping Lemma

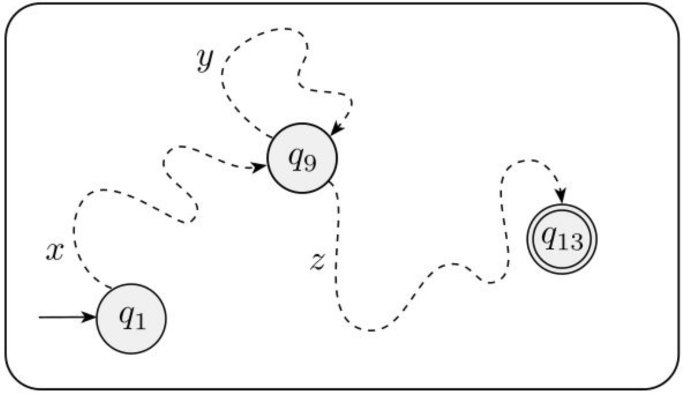
- **Theorem**
  - Pumping lemma If A is a regular language, then there is a number p (the pumping length) where if **s** is any string in **A** of length **at least** p, then s may be divided into three pieces, s = xyz, satisfying the following conditions:
    - for each $i \geq 0$, $xy^i z \in A$,
    - $|y| > 0$, and
    - $|xy| \leq p$.
  - P is called the **pumping length**



Non-Regular Languages
Pumping Lemma

- **Theorem**
  - Pumping lemma If A is a regula
    pumping length) where if **s** is a
    may be divided into three piec
    conditions:
    - for each $i \geq 0$, $xy^i z \in A$,
    - $|y| > 0$, and
    - $\boxed{|xy| \leq p.}$
  - P is called the **pumping length**

P can be considered as the number of states to travel from Q1 to Q9 and **travel only once y path** .



Non-Regular Languages
Pumping Lemma

- **Pumping property**
  - If a language is regular, then it must have the pumping property.
  - If a language does not have the pumping property, then the language is not regular.
- **How to prove languages non-regular using pumping lemma?**
  - Proof by contradiction.
  - Assume that the language is regular.
  - Show that the language does not have the pumping property.
  - Contradiction : Hence, the language has to be non-regular.

Non-Regular Languages
Pumping Lemma

- **Example : B = { 0^n : n ≥ 0 }**
  - Is this language regular ?

Non-Regular Languages
Pumping Lemma

- **Example : B = { 0^n1^n : n ≥ 0 }**
    - Is this regular ?

Non-Regular Languages
Pumping Lemma

- **Example : B = { 0^n1^n : n ≥ 0 }**
  - Prove that the language B is non-regular
    - We assume B is regular and is accepted by DFA with N states.
    - Let's Consider the specific string s = $\mathbf{0^P\ 1^P}$ from the language B
    - Split s =x**y**z according to Pumping Lemma.
      - Examples :
        - 000111
        - 0000011111

Non-Regular Languages
Pumping Lemma

- **Example : B = { $0^n 1^n$: n ≥ 0 }**
  - Prove that the language B is non-regular
    - Remember our string s= x**y**z = **0^P 1^P**
    - Since **|xy| ≤ P**, it follows that **y** is composed entirely of 0's.
      - **For instance** if P=4 : s=00001111 , as **|xy|<4 then xy** must be a substring in 0000 (Fixing a value of P is only for explanation, don't ever fix a value for P)
    - xy ? y what it can be ? : provided that **|y|>0**
    - If we pump for y^2 or y^3.... : we obtain

Non-Regular Languages
Pumping Lemma

- **Example : B = { 0^n1^n : n ≥ 0 }**
  - Prove that the language B is non-regular
    - Remember our string s= x**y**z = $\mathbf{0^P\ 1^P}$
    - Since **|xy| ≤ P**, it follows that **y** is composed entirely of 0's.
    - Let's assume that **|y|=k**
    - If we pump for y^2 or y^3.... : we obtain
      - We just repeat zeros without repeating the 1, Therefore,
      - xy^2z =xyyz=$\mathbf{0^{P+k}\ 1^P}$ does not belong to B because the number of zeros is not equal to the number of 1s, you may do it for i=3,4....
    - This contradicts the Pumping Lemma. Therefore B is not regular

Non-Regular Languages
Pumping Lemma

- **Example : B = { 0^n1^n : n ≥ 0 }**
  - Prove that the language B is non-regular

We cannot find a way to pump/generate more strings which must be in the same language,

of

zeros is not equal to the number of 1s

- This contradicts the Pumping Lemma. Therefore B is not regular

Non-Regular Languages
Pumping Lemma

- **Example : B = { 0^n1^n : n ≥ 0 }**
  - Prove that the language B is non-regular

To efficiently use the pumping lemma :

<u>Find a string</u> that's in the language but you cannot generate more strings from it in the language

of

zeros is not equal to the number of 1s

  - This contradicts the Pumping Lemma. Therefore B is not regular

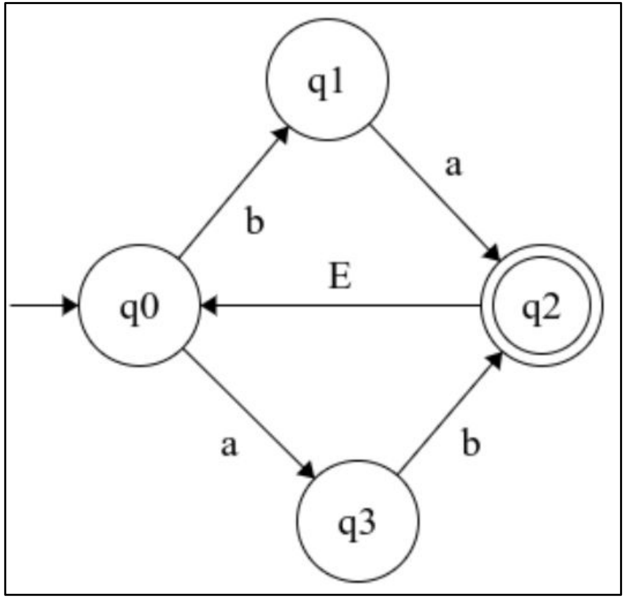
Non-Regular Languages
Pumping Lemma

- **Example : $L = \{ w \mid n_a(w) = n_b(w)$, $n_a(w)$ is the number of occurrences of a in w$\}$**
    - Prove that the language L is non-regular or **Regular**
    - Can we design an NFA/DFA for it ?

Non-Regular Languages
Pumping Lemma

- **Example : L = { w | n_a (w) = n_b (w), n_a(w) is the number of occurrences of a in w}**
  - Prove that the language L is non-regular

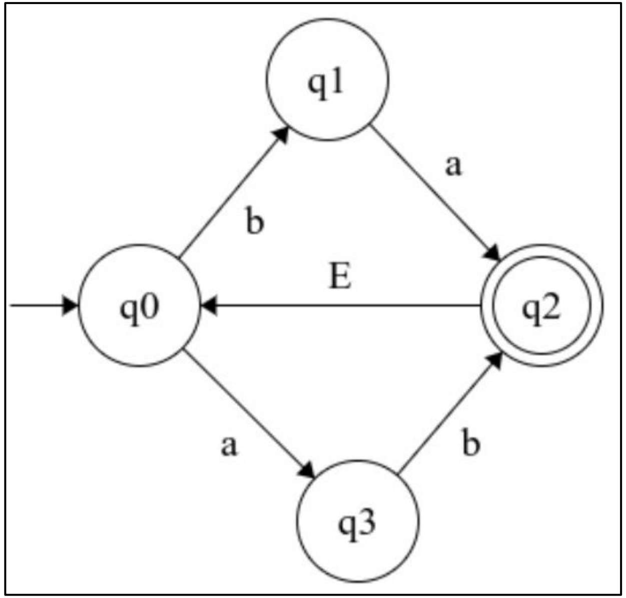
    or **Regular**
  - Can we design an NFA/DFA for it ?



Non-Regular Languages
Pumping Lemma

- **Example : L = { w | n_a (w) = n_b (w), n_a(w) is the number of occurrences of a in w}**
  - Prove that the language L is non-regular

    or **Regular**
  - Can we design an NFA/DFA for it ?
    - What about the following words:
      - bbbaaa
      - bbaaba
    - Can we try a regular expression instead?



Non-Regular Languages
Pumping Lemma

- **Example : L = { w | n_a (w) = n_b (w), n_a(w) is the number of occurrences of a in w}**
  - Prove that the language L is non-regular or **Regular**
  - We use the pumping lemma:
    - Suppose L is regular. Then it must satisfy pumping property.
    - We observe that L is infinite.
    - We consider the pumping length **P**

Non-Regular Languages
Pumping Lemma

- **Example : L = { w | n_a (w) = n_b (w), n_a(w) is the number of occurrences of a in w}**
  - Prove that the language L is non-regular or **Regular**
  - We use the pumping lemma:
    - Suppose L is regular. Then it must satisfy
    - We observe that L is infinite.
    - We consider the pumping length **P**
    - **Let's take the string s =$(ab)^P$ from L (abababab….. ab is repeated P times)**

$s=(ab)^P$

Non-Regular Languages
Pumping Lemma

- **Example : L = { w | n_a (w) = n_b (w), n_a(w) is the number of occurrences of a in w}**
  - Prove that the language L is non-regular or **Regular**
  - We use the pumping lemma:
    - Let's take the string **s =$(ab)^P$** from L (abababab….. ab is repeated **P times**)
    - If s is split into **xyz** such that **|xy| <= P**
      - **xy should be $(ab)^M$ such that M<=P/2** (a, ab, abab, ababa …)
      - Then, **y** can be **ab** or multiple of **ab**
      - Let's try to pump **y**

Non-Regular Languages
Pumping Lemma

- **Example : L = { w | n_a (w) = n_b (w), n_a(w) is the number of occurrences of a in w}**
  - We use the pumping lemma:
    - Let's take the string **s =(ab)^P** from L (abababab..... ab is repeated **P times**)
    - If s is split into **xyz** such that **|xy| <= P**
      - **xy should be (ab)^M such that M<=P/2** (a, ab, abab, ababa ...)
      - Then, **y** can be **ab** or multiple of **ab**,
      - By pumping y or repeating y : x**y²**z, x**y³**z ...
        - We will have the same number of a and b. Therefore, the new generated strings would be part of the language L

Non-Regular Languages
Pumping Lemma

- **Example : L = { w | n_a (w) = n_b (w), n_a(w) is the number of occurrences of a in w}**
  - We use the pumping lemma:
    - Let's take the string **s =(ab)^P** from L (abababab.... ab is repeated **P times**)
    -)

No contraction here ?
Does it mean the language is regular ?

  - We will have the same number of a and b. Therefore, the new generated strings would be part of the language L

Non-Regular Languages
Pumping Lemma

- **Exam… v}**
  - V…
    - … **es**)

It means you picked a bad string .
(ab)*

Pumping lemma can never be used to prove that a language is regular

- We will have the same number of a and b. Therefore, the new generated strings would be part of the language L

Non-Regular Languages Pumping Lemma

- **Example : L = { w | n_a (w) = n_b (w), n_a(w) is the number of occurrences of a in w}**
  - Prove that the language L is non-regular or **Regular**
  - If there is no **Contradiction,** it means you chose a **bad** string ,
    - Other string to take ?

Non-Regular Languages
Pumping Lemma

- **Example : L = { w | n_a (w) = n_b (w), n_a(w) is the number of occurrences of a in w}**
  - Prove that the language L is non-regular or **Regular**
    - Other string to take ?
    - **$s=a^Pb^P$**
    - Solved using the same example.

Non-Regular Languages
Pumping Lemma

- **Prove that {ww | w $\in \Sigma^*$} is a non-regular language?**
  - Use Pumping lemma (not the intuition that we need more memory)
    - What string you would take to arrive to a contradiction
      - Regardless of how you repeat the **y** it won't be part of the language

Non-Regular Languages Closure Property

- **Using property of Regular Languages:**
  - L = { w | n_0 (w) = n_1 (w), n_1(w) is the number of occurrences of 1 in w}
  - B = { 0^n1^n : $n \geq 0$ }
    - **B is non-regular**
    - **B is a subset of L**
    - Therefore : **can we deduce that L is non-regular**

Non-Regular Languages Closure Property

- **Using property of Regular Languages:**
  - L = { w | n_0 (w) = n_1 (w), n_1(w) is the number of occurrences of 1 in w}
  - B = { 0^n1^n : n ≥ 0 }
    - B is non-regular
    - B is a subset of L
    - Therefore : can we deduce that L is non-regular
      - **No, because the superset Σ* of all languages is regular**

Non-Regular Languages

- U

Remember: Closure Rules apply only for :
Regular Operation Regular ⇒ Regular

Non-regular Operation Non-Regular ⇒ We don't know (They may give regular, or even non-regular)

Operations : { complement, Union, Concat, Star, Intersection }

Non-Regular Languages Closure Property

- **Using Closure property of Regular Languages:**
  - L = { w | n_0 (w) = n_1 (w), n_1(w) is the number of occurrences of 1 in w}
  - B = { $a^n b^n$: n ≥ 0 }
  - **C is a language represented by a^*b^***
    - C is regular
  - **L ∩ C = B**

Non-Regular Languages Closure Property

- **Using Closure property of Regular Languages:**
  - L = { w | n_a (w) = n_b (w), n_a(w) is the number of occurrences of a in w}
  - B = { $a^n b^n$: n ≥ 0 }
  - **C is a language represented by a^*b^***
    - C is regular
    - B is already proved as non-regular
    - We assume that L is **regular**
      - **L ∩ C = B : But** B is non-regular, which is a contradiction since the intersection of two regulars must give a regular.
      - **Therefore L is non-regular**

Non-Regular Languages Closure Property

- **Using property of Regular Languages:**
  - What about the language :
    - $D = \{w \mid w = a^m b^n, m \neq n\}$
    - Prove using the closure property of regular languages

Non-Regular Languages Closure Property

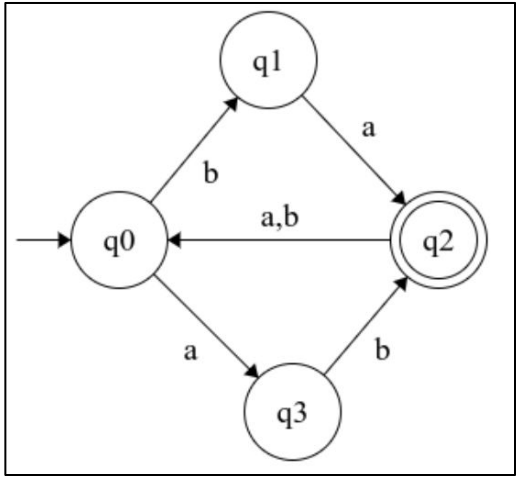
- **Using property of Regular Languages:**
  - What about the language :
    - $D = \{w \mid w = a^m b^n , m \neq n\}$
    - Prove using the closure property of regular languages
      - $B = \{ a^n b^n : n \geq 0 \}$
      - **C is a language represented by a^*b^***
        - C is regular considered as the universal set.
        - B is already proved as non-regular
        - We assume that D is **regular**
        - **Complement (D) over C is B**
          - **If D is regular therefore, B must be regular ! contradiction**

Non-Regular Languages Closure Property

- **Using property of Regular Languages:**
  - Closure Property for Non-regular Languages :
    - D = $\{w \mid w = a^m b^n , m \neq n\}$ (Non-regular)
    - B = $\{ a^n b^n : n \geq 0 \}$ (Non-regular)
    - M = $\{ a^n b^{2n} : n \geq 0 \}$ (Non-regular)
    - D U B = $a^* b^*$ (Regular)
    - Regular U Regular ⇒ **Must** be regular :
    - Non-regular U Regular ⇒ **We don't know (**$\{ a^n b^n : n \geq 0 \}$ U $(a|b)^*$? **)**
    - Non-regular U Non-regular ⇒ **We don't know**
      - *$\{w \mid w = a^m b^n , m \neq n\}$ U $\{ a^n b^n : n \geq 0 \}$ = $a^* b^*$ ⇒ Regular*
      - *$\{ a^n b^n : n \geq 0 \}$ U $\{ a^n b^{2n} : n \geq 0 \}$ ⇒ Non-Regular*

Distinguishable Strings

- **Simplification**
  - Given the following **DFA** machine:
  - Example :
    - *Let's compute the transitions for two strings :*
      - x=***ab***
      - y=***ba***
    - *We observe that they end up at the same state*
      - $\delta^*(q0, x) = \delta^*(q0, y) = q2$



Distinguishable Strings

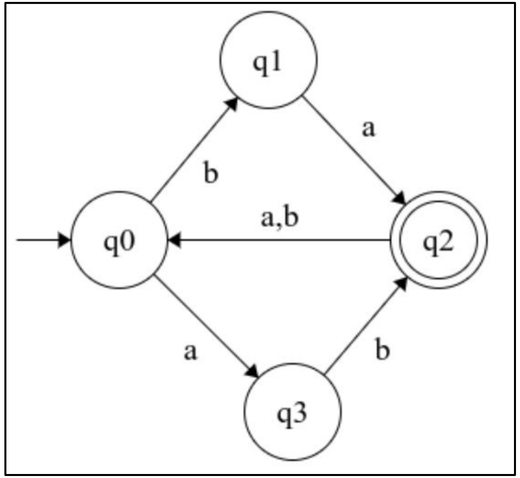
- **Simplification**
  - Give
  - Exar

What happens if we add/append any string z to x and y ?

- *x=ab**ababa***
- *y=ba**ababa***

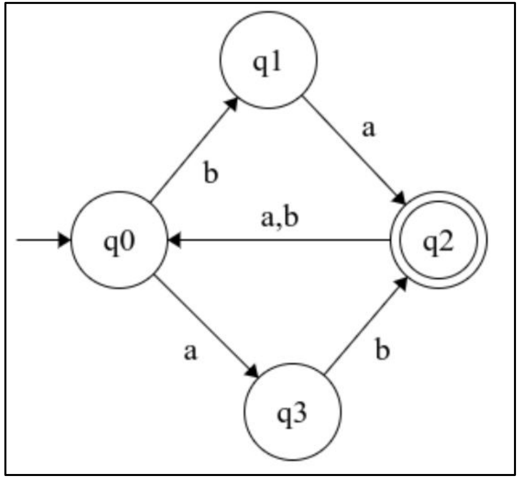
Will they lead to the same state ?
- **It is a must YES for DFA**

- $\delta^*(q0, x) = \delta^*(q0, y) = q2$



Distinguishable Strings

- **Simplification**
  - Given the following **DFA** machine:
  - Example :
    - *Let's take two different strings*
      - x=***a***
      - y=***ba***
    - *We observe that they end up at* ***different states***
      - $\delta^*(q0, x) = q3$
      - $\delta^*(q0, y) = q2$

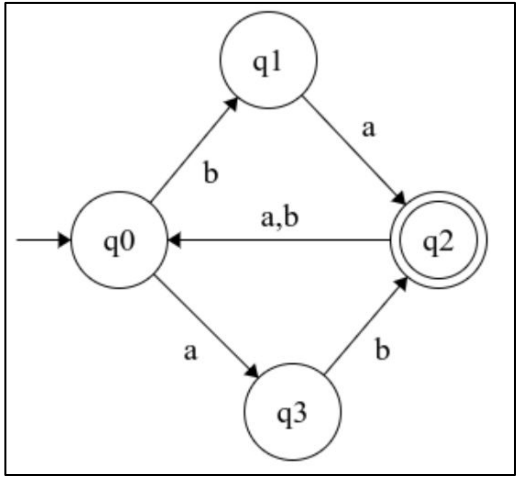


Distinguishable Strings

- **Simplification**
  - Give
  - Exam
    - *y=ba*
    - *We observe that they end up at* ***different states***
      - $\delta^*(q0, x) = q3$
      - $\delta^*(q0, y) = q2$

What happens if we add/append any string z to x and y ?

- *x=a****baba*** $\rightarrow$ ***q2*** $\rightarrow$ ***Accepted***
- *y=ba****baba*** $\rightarrow$ ***q0*** $\rightarrow$ ***Not accepted***



Distinguishable Strings

- **Definition**
  - Given a language L over a finite alphabet Σ, two strings x, y ∈ Σ^* are **suffix distinguishable** with respect to L if **<u>there is</u>** a string **z ∈ Σ^*** such that
    - Exactly one of xz, yz is in L.
      - xz ∈ L and yz ∉ L **Or**
      - xz ∉ L and yz ∈ L
    - We say that z is a **distinguishing suffix** for x, y in L

Distinguishable Strings

- **Definition**
  - Given a language L over a finite alphabet Σ, two strings x, y ∈ Σ^* are **suffix distinguishable** with respect to L if **<u>there is</u>** a string **z ∈ Σ^*** such that
    - Exactly one of xz, yz is in L

      They are different or separate states

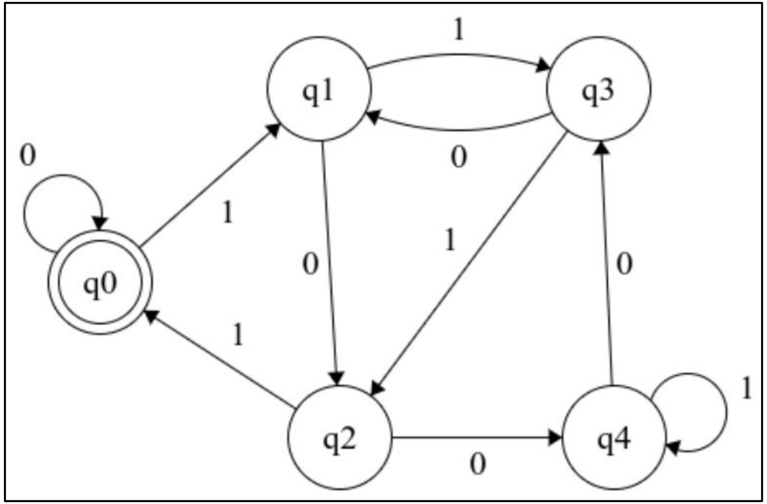
      - xz [illegible] L and yz [illegible] L or
      - xz ∉ L and yz ∈ L
    - We say that z is a **distinguishing suffix** for x, y in L

Distinguishable Strings

- **Lemma :**
  - If L has a distinguishable strings x, y and M = (Q, Σ, δ, s, A) is any DFA that recognizes L
    - then $\delta^*(s, x) \neq \delta^*(s, y)$

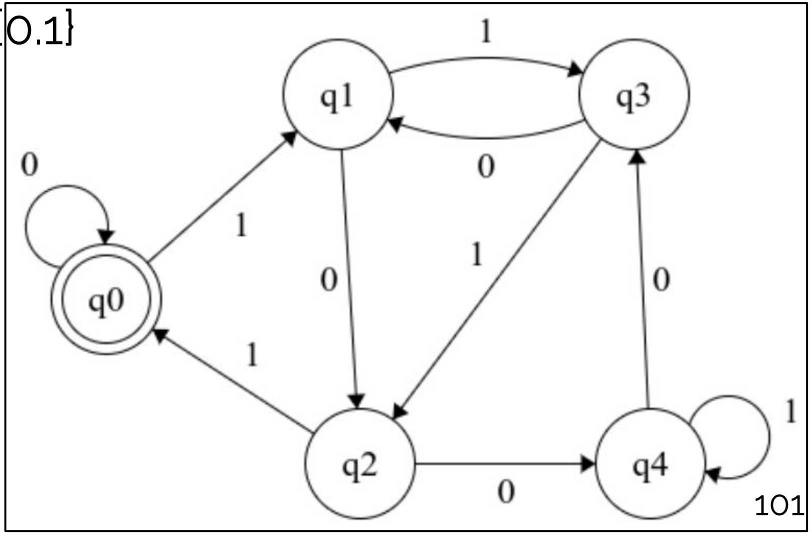
Distinguishable Strings

- **Example:**
  - Given the following DFA machine , give the possible distinguishable strings
    - What's the language for this DFA ?



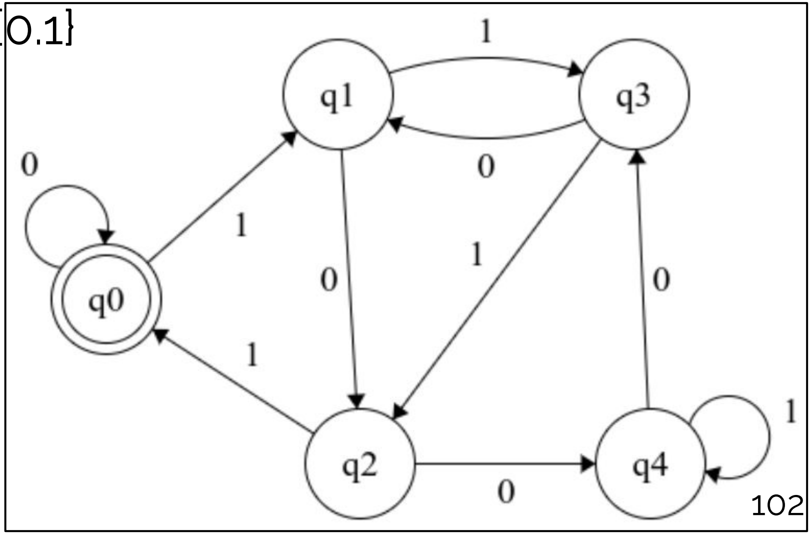
Distinguishable Strings

- **Example:**
  - Given the following DFA machine , give the possible distinguishable strings
    - L = { w | w is divisible by 5 } over Σ={0,1}
    - Possible strings:
      - 0
      - 1
      - 11
      - 10
      - 100



Distinguishable Strings

- **Example:**
  - Given the following DFA machine , give the possible distinguishable strings
    - L = { w | w is divisible by 5 } over Σ={0.1}
    - Possible strings:
      - 0 **(q0)**
      - 1 (**q1)**
      - 11 **(q3)**
      - 10 **(q2)**
      - 100 **(q4**



From the set F, let take any pair of two strings, for example :

- *x=1*
- *y=10*

If we add the string z=01

- ***x=101 → 5 → divisible by 5***
- ***y=1001 → 9 → not divisible by 5***

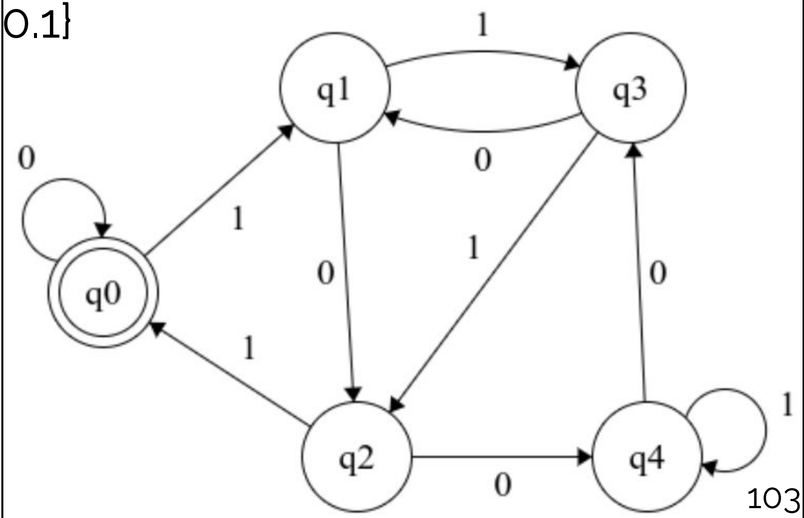
S

-

e the possible distinguishable strings

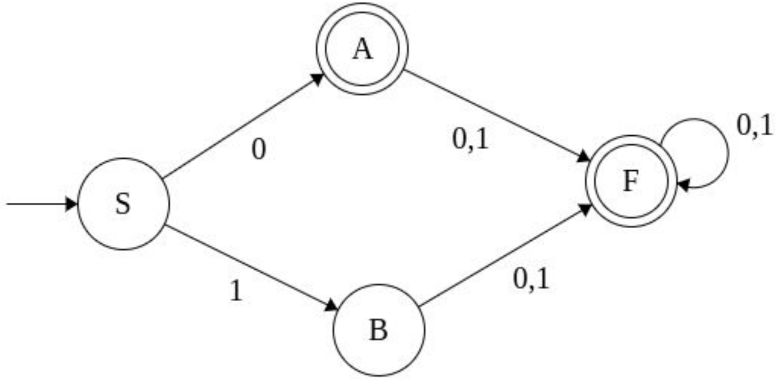
{0.1}

- 0 **(q0)**
- 1 (**q1)**
- 11 **(q3)**
- 10 **(q2)**
- 100 **(q4**



Distinguishable Strings

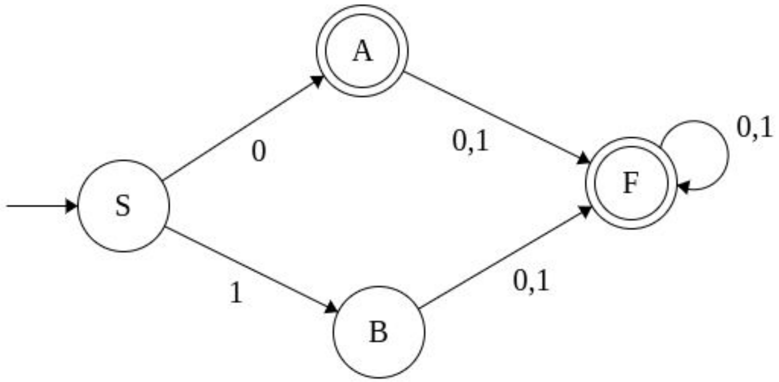
- **Example:**
  - Given the following DFA machine , give the possible distinguishable strings
    - Possible strings:
      -



Good Example given by Abdelhakim, G5

Distinguishable Strings

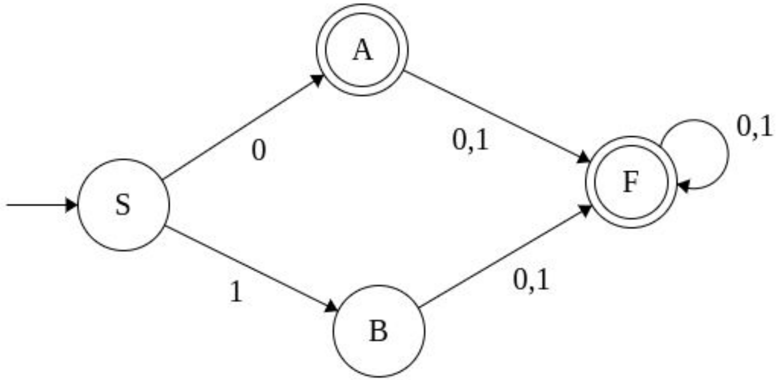
- **Example:**
  - Given the following DFA machine , give the possible distinguishable strings
    - Possible strings:
      - 0
      - 1



Good Example given by Abdelhakim G5

Distinguishable Strings

- **Example:**
  - Given the following DFA machine , give the possible distinguishable strings
    - Possible strings: **Append 00**
      - 0**00**→ **F**
      - 1**00** → **F**



Good Example given by Abdelhakim G5

Non-Regular Languages
Fooling Sets

- **Fooling Set :**
  - **Definition:**
    - Let L be a language. A set of strings **F** is a fooling set for L if every pair of distinct strings in F **is distinguishable with respect to L**
  - **Simplification:**
    - If F = { x, y, c}
      - x and y must be distinguishable with respect to L
        - **There is** a string z such that strictly either xz or yz belong to L
      - y and c ….
      - x and c …

Non-Regular Languages
Fooling Sets

- **Theorem :**
  - Let L be a language and let **F be a fooling** set for L. No DFA M can recognize L if it has less than |F| states.
  - If **|F | is infinite** then L cannot be regular = is a **non-regular** language

Non-Regular Languages
Fooling Sets

- **Myhill-Nerode Theorem :**
  - Let L be any language. Then
    - If L is not regular then there is an infinite fooling set for L.
    - If L is regular then there is a fooling set F of size k where k is the smallest number of states of a DFA that accepts L.

Non-Regular Languages
Fooling Sets

- **Example : B = { 0^n1^n : n ≥ 0 }**
  - Prove that the language B is non-regular

Non-Regular Languages
Fooling Sets

- **Example : B = { 0^n1^n : n ≥ 0 }**
  - Prove that the language B is non-regular
    - Let's assume that **F = { 0^* }** as the Fooling set of B
      - If we consider two strings s_1,s_2 as 0^i and 0^j respectively from the the set **F** such that i ≠ j
      - If we consider the string **$z=1^i$** then (111..... i times):
        - $s_1z=0^i1^i$ which is from language B
        - $s_2z=0^j1^i$ does not belong to language B because i ≠ j

Non-Regular Languages
Fooling Sets

- **Example : B = { 0^n1^n : n ≥ 0 }**
  - Prove that the language B is non-regular
    - Let's assume that F = { 0^* } as the fooling set of B
      - For any two different values i and j (infinite possibilities)
      - Whilst s_i and s_j are **distinguishable**, they lead to **different states**
        - **How many states do we need ? Infinite number**
        - **Therefore : we cannot have a finite automaton for this language**

Next ?

- **Automaton**
  - Machine that would accept the language $\mathbf{a^n b^n}$
  - **Time to create a machine with some memory ?**

Course Content

- **Introduction**
    - Complexity theory, Computability theory, Mathematical notions, Types of Proofs
- **Automata theory**

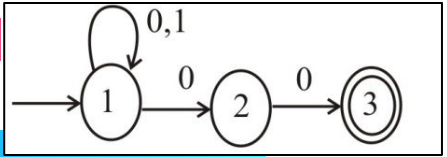
5 weeks

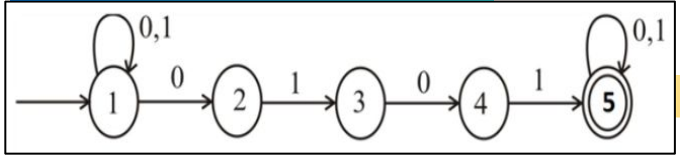
    - Regular Languages : Finite Automata, Non-determinism, Regular Expressions, nonregular languages.
    - Context-free languages : Grammars, Pushdown automata
- **Computability theory**
    - Turing machines, recursively enumerable and recursive languages
    - Church-Turing thesis
    - Decidability
    - Reducibility
- **Complexity Theory**
    - Complexity of algorithms and of problems
    - Complexity classes P, NP, PSPACE
    - Polynomial-time reduction
    - NP-Completeness and Cook's theorem + PSPACE-Completeness

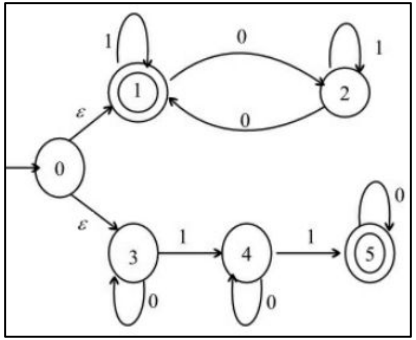
Solutions : NFA TD3

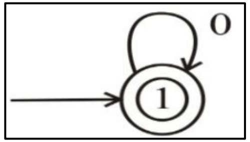
- **Ex 1**

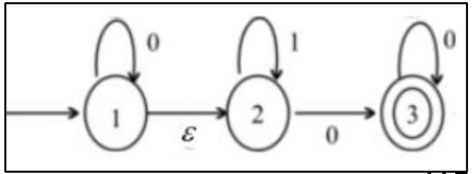
1. **The language {0} with two states**
2. **The language {w| w contains the substring 0101 (i.e., w = x0101y for some x and y)} with five states**
3. **The language {w| w contains an even number of 0s, or contains exactly two 1s} with six states**
4. **The language 0^* with one state**
5. **The language 0^* 1^* 0^+ with three states**
6. Let Σ = {a, b, c} and let L = { w ∈ Σ^* | some character in Σ appears at most twice in w }







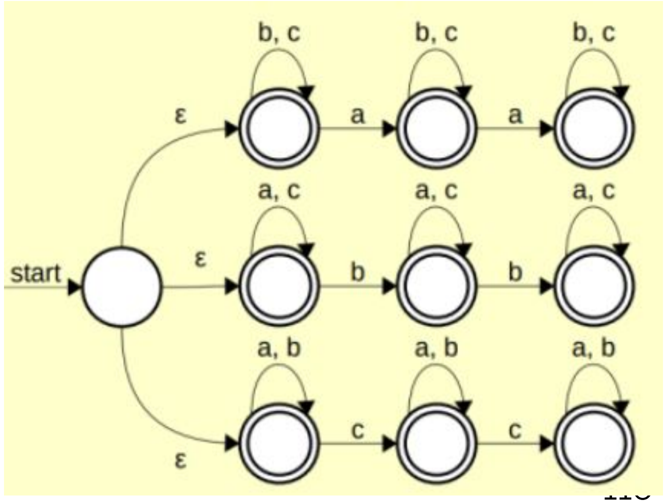




Solutions : NFA TD3

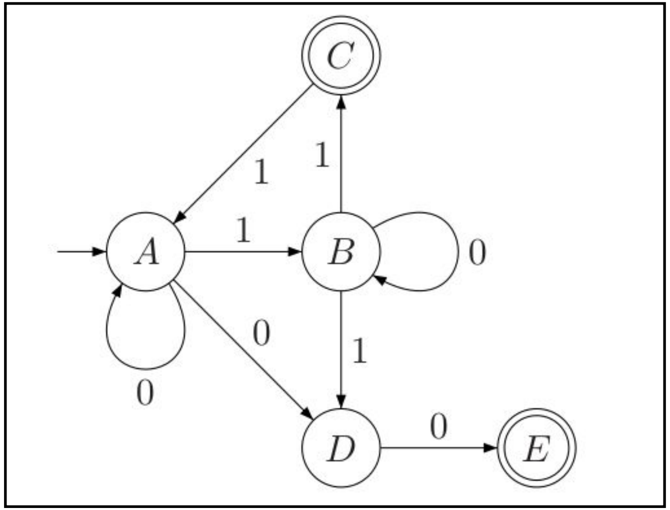
- **Ex 1**

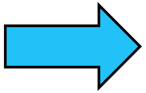
1. The language {0} with two states
2. The language {w| w contains the substring 0101 (i.e., w = x0101y for some x and y)} with five states
3. The language {w| w contains an even number of 0s, or contains exactly two 1s} with six states
4. The language 0^* with one state
5. The language $0^* 1^* 0^+$ with three states
6. **Let $\Sigma = \{a, b, c\}$ and let $L = \{ w \in \Sigma^* \mid$ some character in Σ appears at most twice in $w \}$**

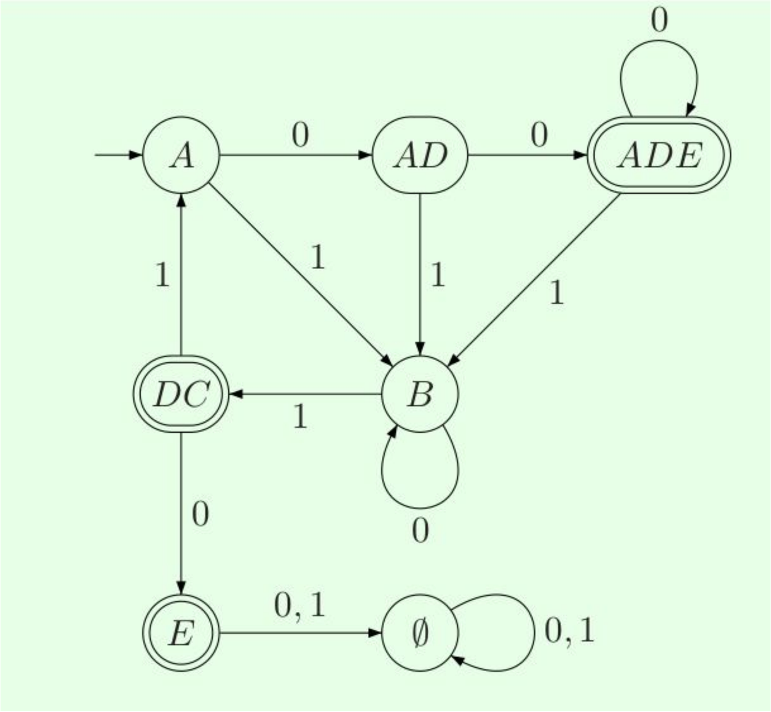


Solutions : NFA TD3

- **Ex 2 : Convert NFA to DFA**







Solutions : NFA TD3

- **Ex 3 : Minimize DFA**

Two sets:

- Set 1 : {p , q , r ,t }
- Set 2: { s } (no splitting needed)

Checking Set 1 for the Equivalence:

(p **vs.** q):

- a → (r, s) → (Set 1, Set 2) (not equivalent)
- b (no need as they are not equivalent

(p **vs.** r)

- a → (r, s) → (Set 1, Set 2) (not equivalent)
- b (no need as they are not equivalent

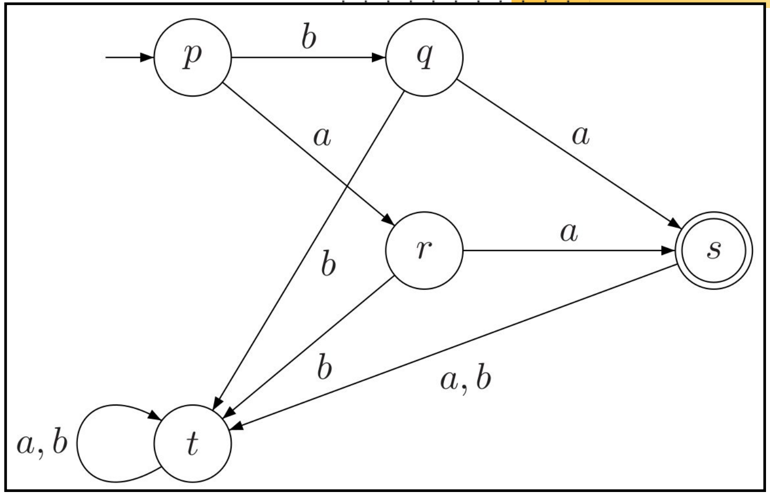
(p **vs.** t)

- a → (r, t) → (Set 1, Set 1) (Equivalent for a)
- b → (q, t) → (Set 1, Set 1) (Equivalent for b)

(q vs. t) no need as they are not equivalent (since p== t whilst q not equivalent to p

(q **vs.** r)

- a → (s, s) → (Set 2, Set 2) (Equivalent for a)
- b → (t, t) → (Set 1, Set 1) (Equivalent for b)



P and T are equivalent

Q and R are equivalent

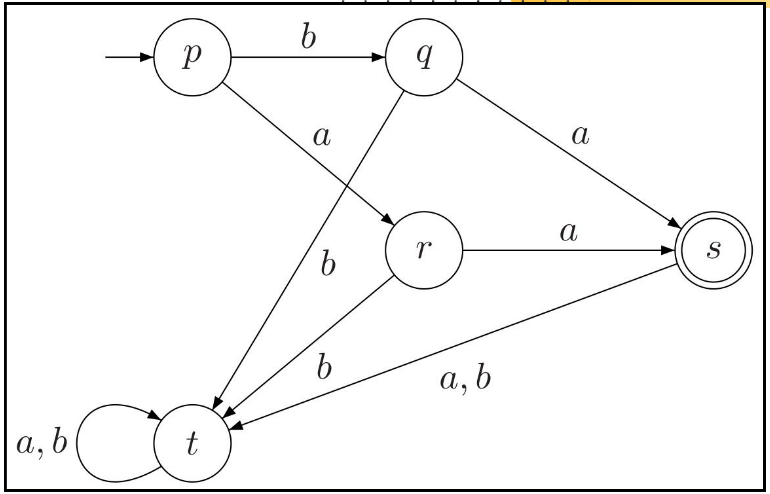
Solutions : NFA TD3

- ## Ex 3 : Minimize DFA

Two sets:
- Set 1 : {p , q , r ,t }
- Set 2: { s } (no splitting needed)

Newer Sets
- Set 1 : {p , t }
- Set 2 : {q , r }
- Set 3 :{ s } (no splitting needed)

Checking Set 1 for the Equivalence:
(p **vs.** t):
- We did before ? but on different sets we have to do it again on the newer sets



Solutions : NFA TD3

- ## Ex 3 : Minimize DFA

Two sets:

- Set 1 : {p , q , r ,t }
- Set 2: { s } (no splitting needed)

Newer Sets

- **Set 1 : {p , t }**
- **Set 2 : {q , r }**
- **Set 3 :{ s } (no splitting needed)**

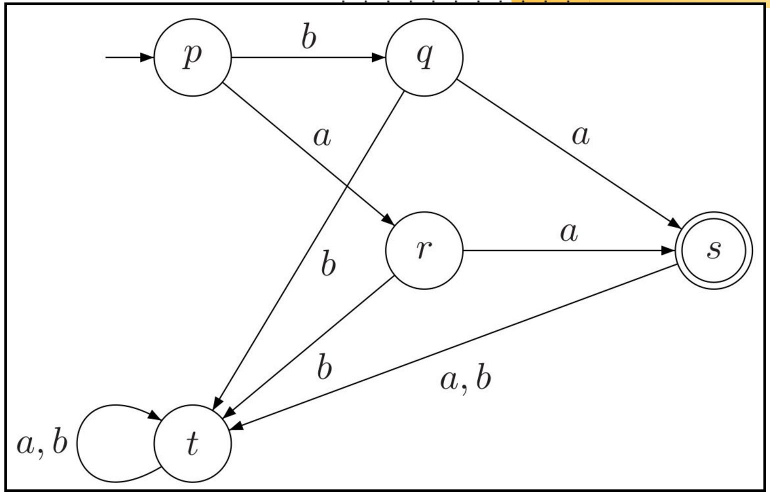
Checking Set 1 for the Equivalence:

(p **vs.** t):

- a → q, t → set 2, Set 1 : Not equivalent

(q **vs.** r):

- a → s, s → set 3, Set 3 : Equivalent
- b → t, t → set 1, Set 1 : Equivalent



Q and R are equivalent

Solutions : NFA TD3

- **Ex 3 : Minimize DFA**

Two sets:

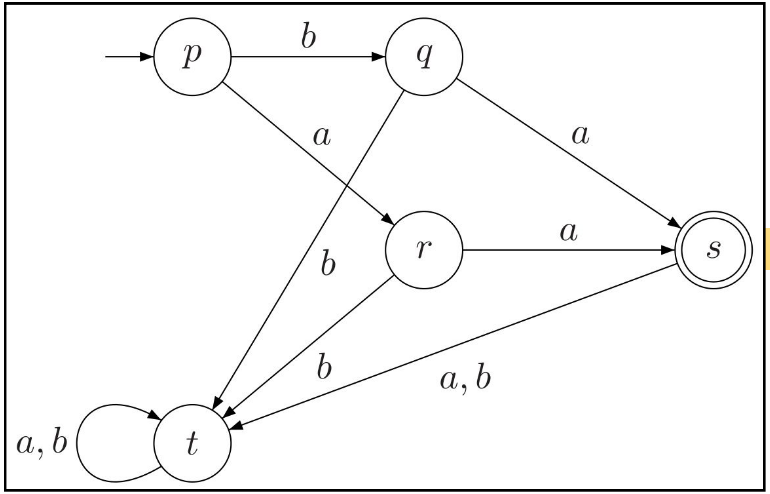
- Set 1 : {p , q , r ,t }
- Set 2: { s } (no splitting needed)

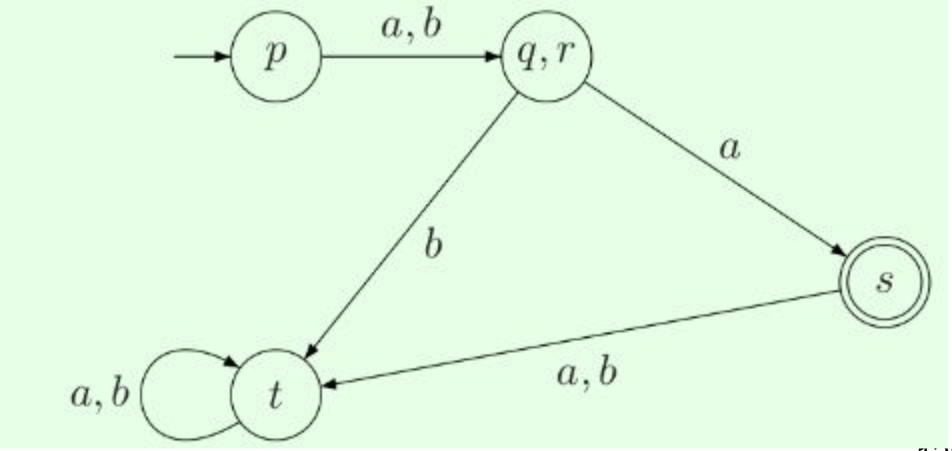
ThreeSets

- Set 1 : {p , t }
- Set 2 : {q , r }
- Set 3 :{ s } (no splitting needed)

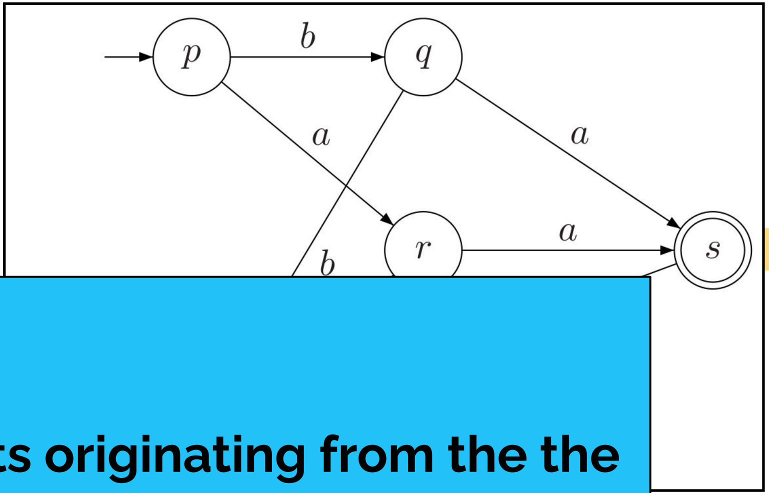
Newer Sets

- **Set 1 : {p }**
- **Set 1 : {t }**
- **Set 2 : {q , r }**
- **Set 3 :{ s } (no splitting needed)**





Solutions : NFA TD3



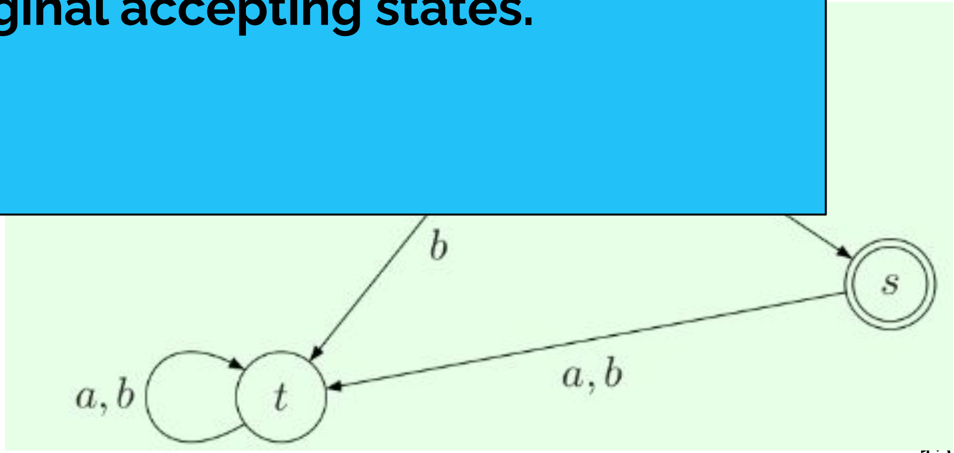
- **Ex 3 : Minimiz**

Two sets:
- Set 1 : {p , q
- Set 2: { s } (

ThreeSets
- Set 1 : {p , t }
- Set 2 : {q , r }
- Set 3 :{ s } (

Newer Sets
- **Set 1 : {p }**
- **Set 1 : {t }**
- **Set 2 : {q , r }**
- **Set 3 :{ s } (no splitting needed)**

Accepting states, all subsets originating from the the set of original accepting states.



Palindromes

The sentence :

WAS IT A CAT I SAW

How many possible ways to read this sentence
We can read at any direction :
UP, LEFT, RIGHT, DOWN.

```
            W
          W A W
        W A S A W
      W A S I S A W
    W A S I T I S A W
  W A S I T A T I S A W
W A S I T A C A T I S A W
  W A S I T A T I S A W
    W A S I T I S A W
      W A S I S A W
        W A S A W
          W A W
            W
```