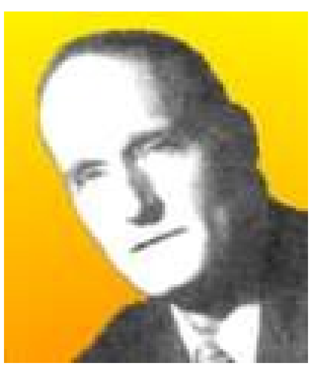
jmsamaniego@uplb.edu.ph

CMSC 141
Finite automata and regular languages

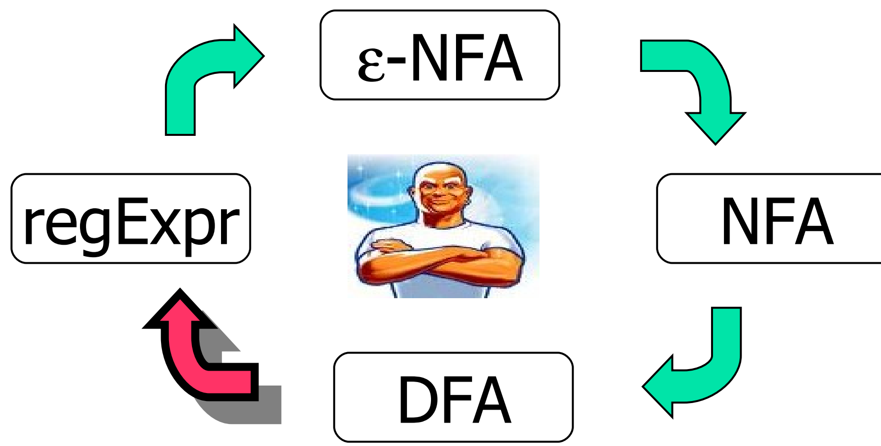
- We had constructive proofs of Kleene’s Theorem:

Finite Automata = Regular Expressions

Finite automata and regular expressions describe exactly the same class of languages – the class of **regular languages**



Stephen Kleene
1909-1994



What's next?

- Minimization of finite automata and simplification of regular expressions
- Finite automata with output (transducers, or Moore and Mealy machines)
- Probabilistic finite automata – related to Markov chains in probability theory
- Regular grammars
- Closure properties of regular sets
- Pumping Lemma for regular sets and non-regular languages

jmsamaniego@uplb.edu.ph

Simplification of regular expressions

(Proofs left as exercises)

jmsamaniego@uplb.edu.ph

- **Identity elements for union and concatenation**

  $\varnothing + x = x + \varnothing = x$ and $\varepsilon x = x\varepsilon = x$

- **Annihilation element for concatenation**

  $\varnothing x = x\varnothing = \varnothing$

- **Commutativity of union**

  $x+y = y+x$

- **Associativity of union, concatenation**

  $(x+y)+z = x+(y+z)$ and $(xy)z = x(yz)$

More identities

(Proofs left as exercises)

- **Distributive properties**

  $x(y+z) = xy + xz$ and $(x+y)z = xz + yz$

  but $(x+y)^* \neq x^*+y^*$ in general ← **why?**

- **Idempotency of union and Kleene closure**

  $x+x = x$ and $(x^*)^* = x^*$ and $(x^+)^+ = x^+$

jmsamaniego@uplb.edu.ph

More identities

(Proofs left as exercises)

- **Absorption property**

  If $x \subseteq y$ then $x+y = y$

- **Kleene star properties**

  $\varepsilon^* = \varepsilon^+ = \varepsilon$ and $\varnothing^* = \varepsilon$

  $(x^*y^*)^* = (x+y)^*$ ← **why?**

jmsamaniego@uplb.edu.ph

Minimization of finite automata

- Idea is to identify pairs of states that are essentially the same, or **indistinguishable**, and merge them into a single state
- Easy to do with a graphical tool like JFLAP in which we can drag the states around
- But how does JFLAP's DFA minimization algorithm work?

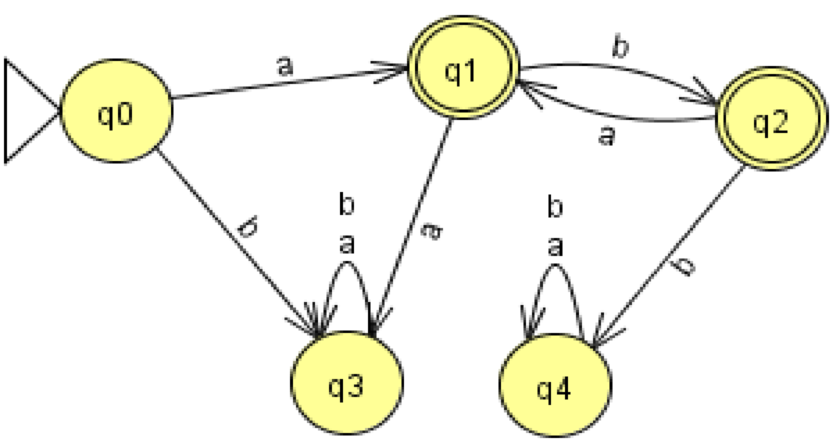
jmsamaniego@uplb.edu.ph

Distinguishable states

a recursive definition

- States p and q are **distinguishable** if
  - one is a final state and the other is a non-final state, or
  - there is some string $x \in \Sigma^*$, such that $\delta(p,x)$ and $\delta(q,x)$ are distinguishable

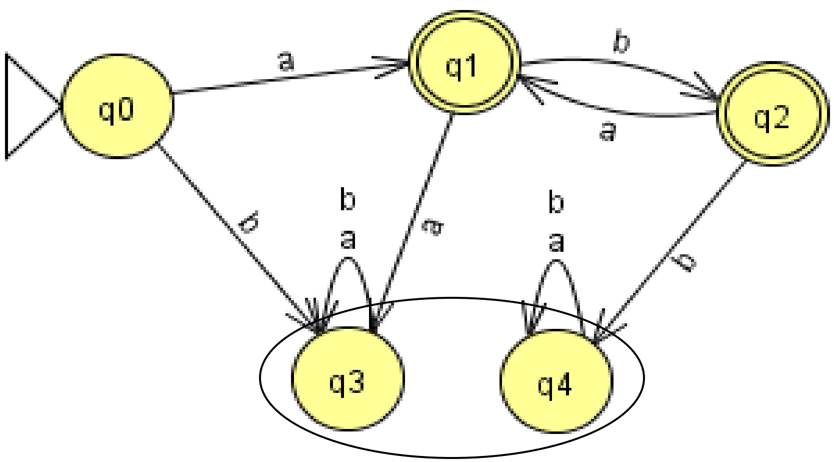
ianiego@uplb.edu.ph



	q_0	q_1	q_2	q_3	q_4
q_0					
q_1	x				
q_2	x				
q_3		x	x		
q_4		x	x		

Minimization of a DFA

jmsamaniego@uplb.edu.ph



	q_0	q_1	q_2	q_3	q_4
q_0					
q_1	x				
q_2	x	x			
q_3	x	x	x		
q_4	x	x	x		

States q1 and q2 are distinguishable:

δ(q1,a) = q3 (a non-final state)

δ(q2,a) = q1 (a final state)

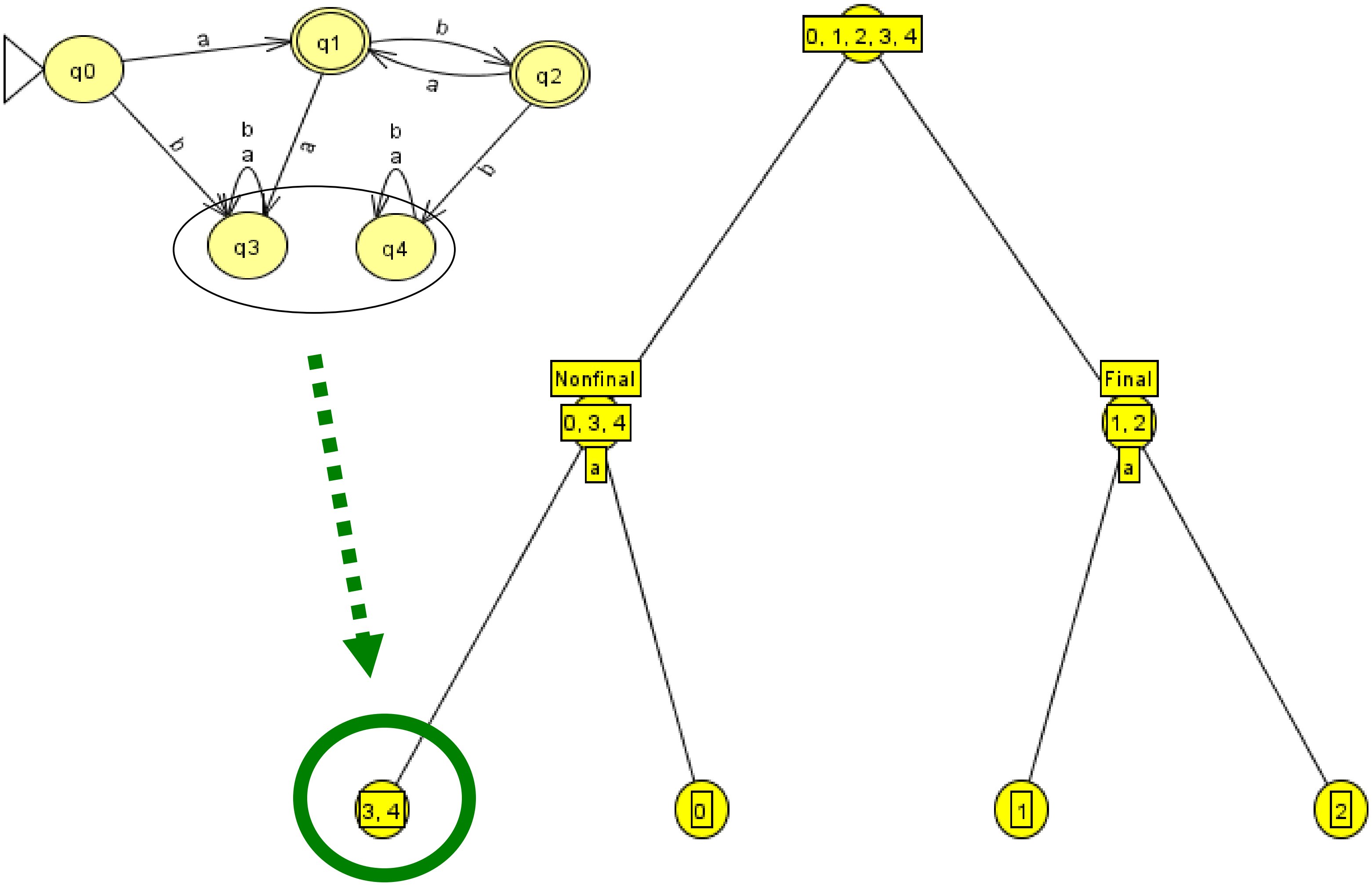
States q3 and q4 are indistinguishable:

for any string x, δ(q3,x) and δ(q4,x) are both non-final states

JFLAP minimizes a DFA

by building a partition tree instead of a matrix

jmsamaniego@uplb.edu.ph



Regular grammars

jmsamaniego@uplb.edu.ph

- Grammars are rule-based systems for describing languages
- Example: the **regular grammar** below consists of a single variable { **S** }, two terminals **{ rose**, **red }** and two production rules

$$\{ \quad \mathbf{S} \rightarrow \mathbf{rose}, \quad \mathbf{S} \rightarrow \mathbf{red}\ \mathbf{S} \quad \}$$

- This grammar generates the language

{ rose, **red rose**, **red red rose**, **red red red rose**, … **}**

Regular grammars

formal definition

jmsamaniego@uplb.edu.ph

- A **regular grammar** is a 4-tuple (V,T,P,S) where
  - V is a finite set of variables
  - T is a finite set of terminal symbols = Σ
  - P is a finite set of production rules, each of the form

    <variable> → <terminal>* or

    <variable> → <terminal>* <variable>

    (right hand side of each rule contains at most one variable at the right end)
  - S is the start variable, S∈V

{ S → rose , S → red S } = S → rose | red S

Derivations

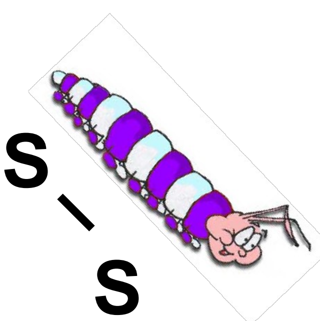
- A grammar **G** is said to generate a string x, if x can be derived from the start variable **S**, by a finite sequence of variable replacements based on the production rules
- Ex: A **linear derivation** and **parse tree** of "**red red rose**"

S → **red S**

→ **red red S**

→ **red red rose**

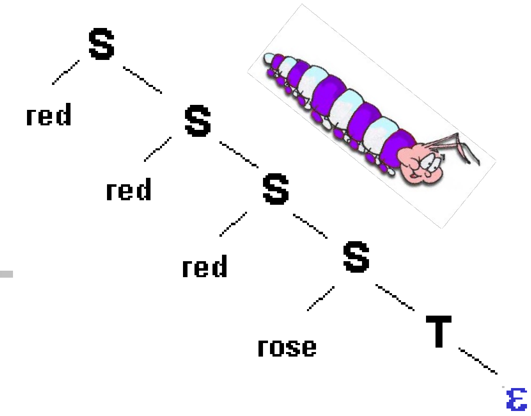
S
red S
red S
rose



- Note that **L(G) = (red)* rose**

jmsamaniego@uplb.edu.ph

Regular grammars = FA



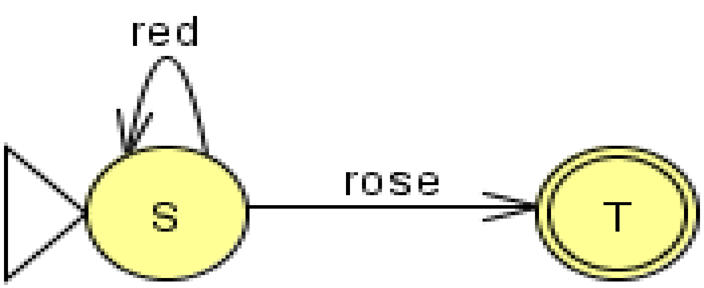
jmsamaniego@uplb.edu.ph

Idea of proof

- FA **states** are **variables** in the regular grammar
- The **start state** is the **start variable**
- **$\delta(X, a) = Y$** if and only if the rule **$X \rightarrow aY$** is present
- **X is a final state** if and only if the rule **$X \rightarrow \varepsilon$** is present

S → red S | rose T

T → ε



Closure properties

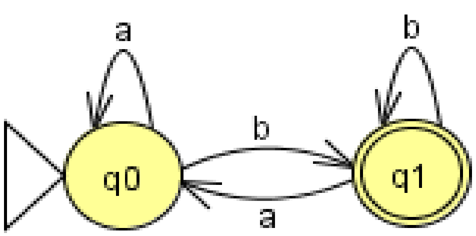
jmsamaniego@uplb.edu.ph

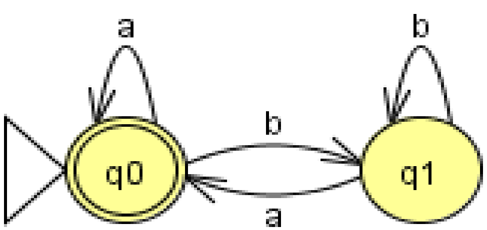
- The **set of regular languages is closed** under **union**, **concatenation**, and **Kleene star**
  - Proof follows immediately from the definition of regular expressions
  - If R and S are regular languages, then they can be described by some regular expressions **r** and **s**
  - **r+s**, **rs**, and **r*** are also regular expressions that describe the regular languages $R \cup S$, RS and R^*

Closure under complementation

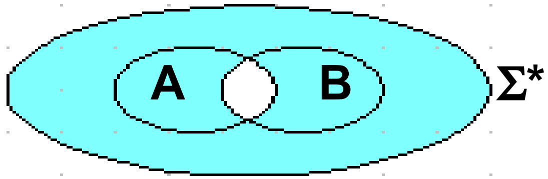
- The **set of regular languages is closed** under **complementation**
- Proof is by FA construction

Build the DFA for L, then negate the status of all the states (all final states are made non-final, and all non-final states are made final). The resulting DFA is the DFA for $L^c = \Sigma^* - L$.





jmsamaniego@uplb.edu.ph



Closure under intersection

- The **set of regular languages is closed** under **intersection**
- Proof is by De Morgan's Law

  $(A \cap B)^c = A^c \cup B^c$ and hence, $A \cap B = (A^c \cup B^c)^c$

  If A and B are regular languages, then so are A^c, B^c, $A^c \cup B^c$, and $(A^c \cup B^c)^c$ by the closure properties we have shown.
- While this proof is valid, actual construction of the FA or the regular expression is a long and complex process. Is there a more direct constructive proof?

jmsamaniego@uplb.edu.ph

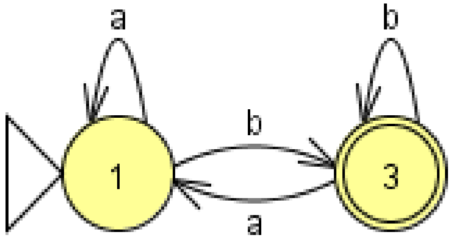
A more efficient constructive proof for closure under intersection

jmsamaniego@uplb.edu.ph

- Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ be the DFA for L_1 and L_2.
- Construct a new DFA $M = (Q, \Sigma, \delta, q_0, F)$ where
  - $Q = Q_1 \times Q_2$ (Cartesian product)
  - $q_0 = (q_1,q_2)$
  - $\delta((p,q),a) = (\delta_1(p,a), \delta_2(q,a))$ for all $p \in Q_1$, $q \in Q_2$, $a \in \Sigma$
  - $(p,q) \in F$ if and only if $p \in F_1$ and $q \in F_2$

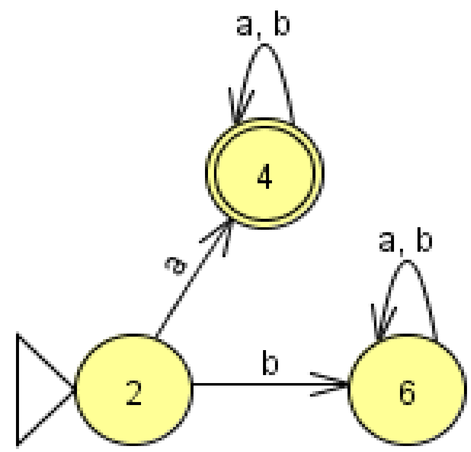
That Greek algorithm deserves an illustration

jmsamaniego@uplb.edu.ph



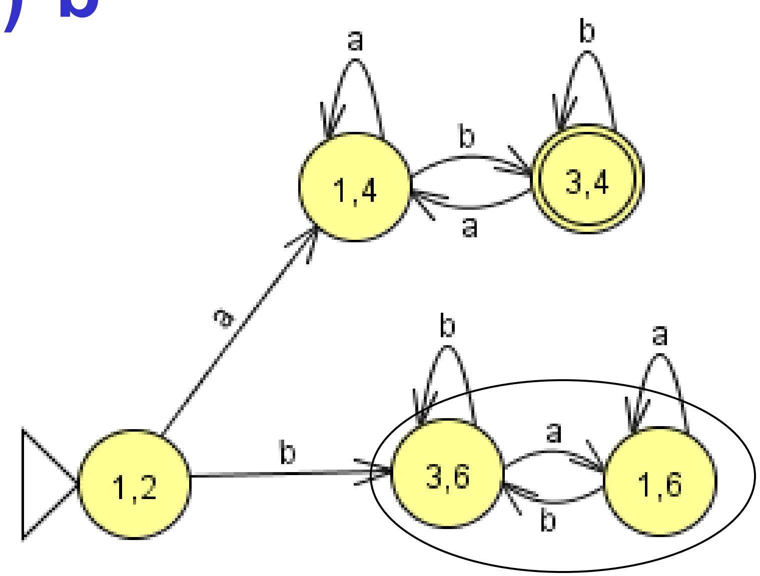
(a+b)*b

∩



a(a+b)*

=



a(a+b)*b

String substitutions

also known as homomorphisms

jmsamaniego@uplb.edu.ph

- Let Σ_1 and Σ_2 be two alphabets.

  A homomorphism is a function h: $\Sigma_1 \rightarrow \Sigma_2^*$

  The domain of h can also be extended to strings or languages, where $h(a_1a_2\ldots a_n) = h(a_1)\, h(a_2)\ldots\, h(a_n)$

  and $h(L) = \{ h(x): x \in L \}$.

- Ex: Let h(**a**) = **0** and h(**b**) = **10**

  If L = { **a**, **ab**, **aba**, **abab**, …} then

  h(L) = { h(**a**), h(**ab**), h(**aba**), h(**abab**), …}

  = { **0**, **010**, **0100**, **010010**, …}

jmsamaniego@uplb.edu.ph

Closure under string substitutions

- **If L is a regular language and h is any homomorphism for L, then h(L) is also a regular language.**
- Proof idea: Apply the homomorphism on the regular expression for L.
- **Example:** Suppose we have the language

  L = a(ba)*(b+ε) and we have the

  homomorphism **h(a) = 0, h(b) = 10**.

  Then **h(L)** is regular and is given by **0(100)*(10+ε)**.

Other closure properties

- Is the set of regular languages closed under
  - string reversal?

    where reverse(L) = { reverse(x) : $x \in L$ }
  - inverse homomorphisms?

    where $h^{-1}(L)$ = { x : h(x) = L }

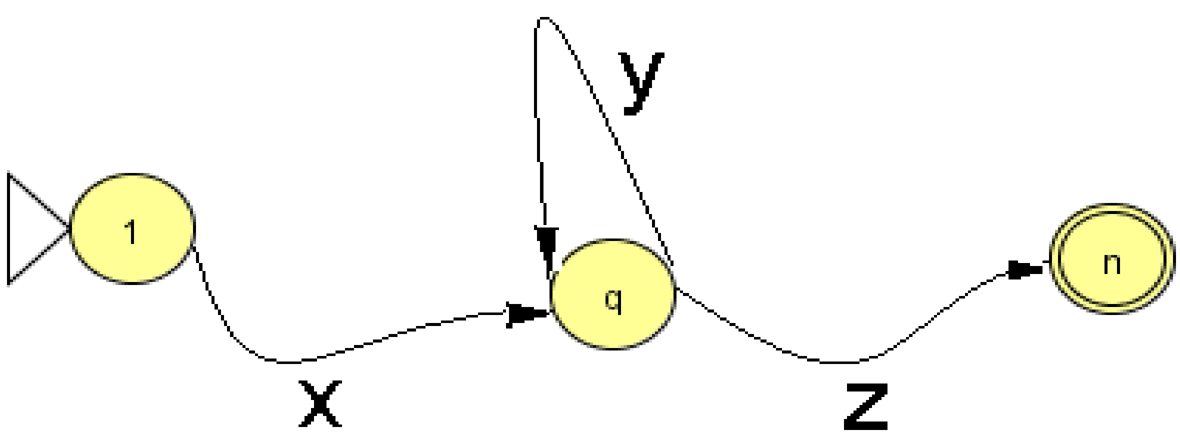
jmsamaniego@uplb.edu.ph

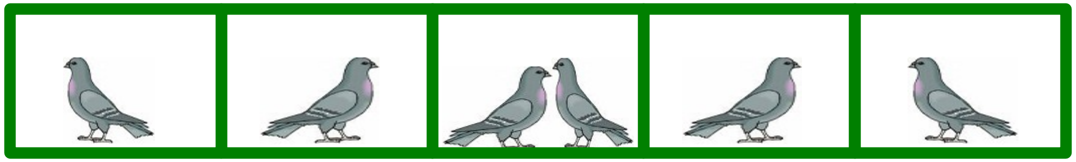
Pumping lemma
for regular languages

jmsamaniego@uplb.edu.ph

- **Short version**: If L is an infinite regular language, and if w is a sufficiently long string in L, then w contains a non-empty substring y that can be repeated (or "pumped") and preserves membership in L, i.e., w can be represented as **w = xyz** such that

  xz, xyz, xyyz, xyyyz,..., xy^jz, ... are all strings in L

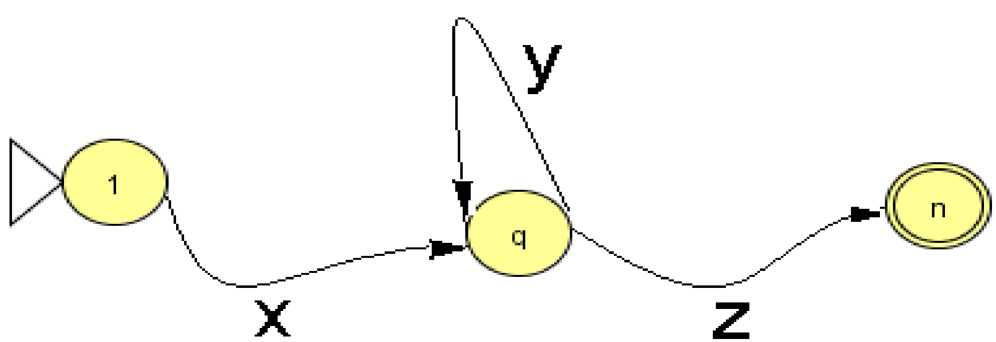




Proof of the pumping lemma

- Pigeonhole principle: If there are more pigeons than pigeonholes, then at least one pigeonhole will contain more than one pigeon.
- If the minimum-state DFA for L has n states, and a string w in L has n or more symbols, then some state q must be revisited. The non-empty substring between these two visits is the string y that we pump. Hence, xy^jz are all in L, for all j = 0, 1, 2, ...

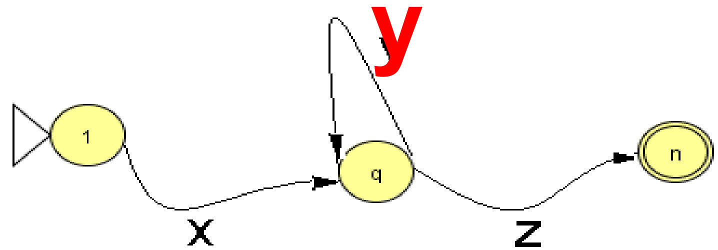
if w = xyz $\in$ L
then xy*z $\subseteq$ L



jmsamaniego@uplb.edu.ph

Pumping lemma

for regular languages



- **Formal Greek version:**

$\forall$ infinite regular languages L,

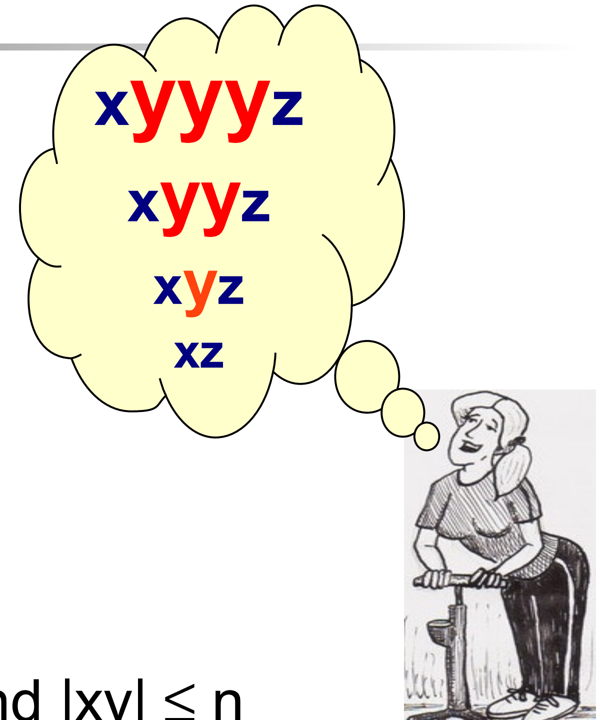
$\exists$ an integer $n > 0$,

$\forall$ strings $w \in \Sigma^*$ with $|w| \geq n$,

$\exists$ a partition of $w = xyz$, with $y \neq \varepsilon$ and $|xy| \leq n$

$\forall$ $j = 0, 1, 2, 3, \ldots$

$$xy^jz \in L$$



jmsamaniego@uplb.edu.ph

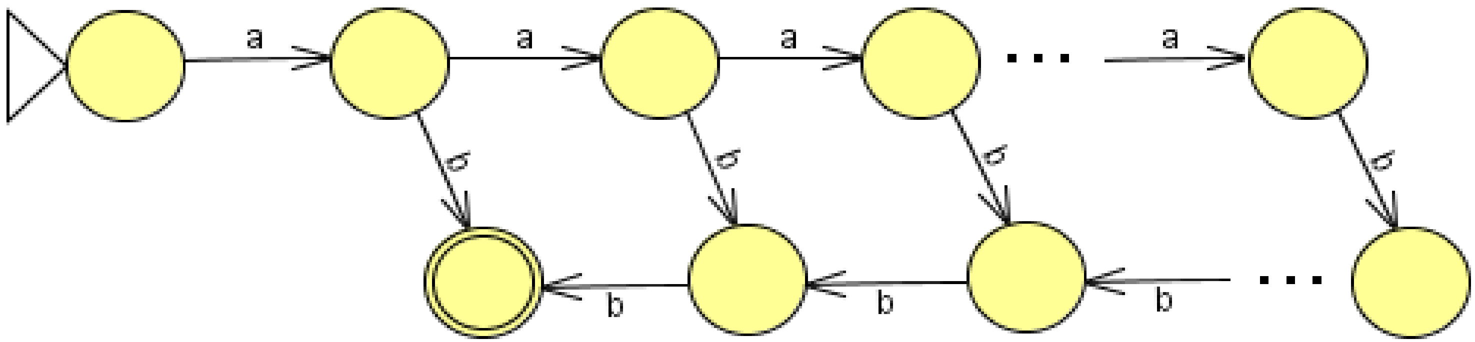
Are all languages regular?

jmsamaniego@uplb.edu.ph

- No, some sets of strings are too complex to be described by FA (or regular expressions or regular grammars).
- Example of a non-regular language?
- Try L $= \{ a^n b^n : n > 0 \}$

  $= \{ ab, aabb, aaabbb, aaaabbbb, \ldots \}$
- Or L' = { (), (()), ((())), (((()))), ... }
- But this doesn't look too complex. Why isn't there any DFA for this simple language?

A candidate DFA for $L = \{ a^n b^n : n>0 \}$?



Why can't this be a valid DFA for L = { ab, aabb, aaabbb, ...}?

jmsamaniego@uplb.edu.ph

The language $L = \{ a^n b^n : n>0 \}$ is non-regular

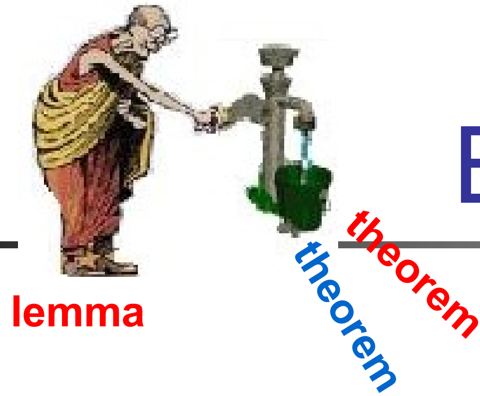
jmsamaniego@uplb.edu.ph

- Just because we were unable to come up with a DFA for L doesn't mean nobody else couldn't. Perhaps we just didn't try hard enough...
- For a real proof of the non-regularity of L, the Pumping Lemma can help us.
- We use the Pumping Lemma to prove our claim via an indirect proof:
  - PL: If L is infinite and regular then ...
  - Indirect proof. We know that L is infinite. Now suppose L is regular. **Then ... (try to reach a contradiction).** Because of the contradiction, L cannot be regular.

Proof using the pumping lemma

jmsamaniego@uplb.edu.ph

- Suppose the language L = { $a^n b^n$: n>0 } is regular. Because it is also infinite, then it must satisfy the pumping lemma.
- Without loss of generality, suppose w = **aaaaabbbbb** is long enough. (If not, try something longer).
- We must be able to partition w into **x(y)z** (with $y \neq \varepsilon$) so that **x(y^j)z** is always in L for any j, j=0,1,2,...
  - Try having all a's for y, e.g., **aa(aaa)bbbbb**
  - Try having all b's for y, e.g., **aaaaa(bb)bbb**
  - Try having a mix of a's and b's for **y**, e.g., **aa(aaabb)bbb**
- In any of these cases, pumping **y** gives strings that are not in L.
- Because finding the substring y is impossible, **L cannot be regular.**



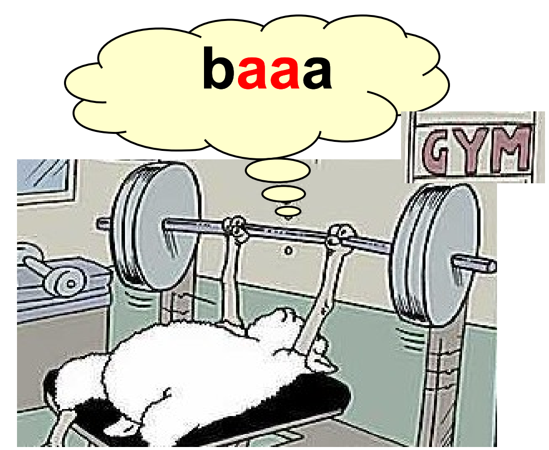
Extra notes

jmsamaniego@uplb.edu.ph

- The **Pumping Lemma** is a theorem, but we call it a **lemma** (an intermediate key **theorem**) because we use it to prove theorems about the **non-regularity** of several languages.
- Note that we cannot use the Pumping Lemma to prove that a certain language is regular, since satisfying the lemma is a necessary but not sufficient condition, for membership in the class of regular languages.
- We do not need to use the pumping lemma every time we need to prove the non-regularity of some language L. Sometimes, the closure properties come in handy.

Exercises

- Prove carefully using the **string substitution closure property**, that { (), (()), ((())), (((()))), ... } is non-regular.
- Prove using the **pumping lemma** that the language L = $\{a^p : p \text{ is prime}\}$ is non-regular.
- Prove that the language L' = $\{a^p : p$ is **not** a prime$\}$ is also non-regular.



jmsamaniego@uplb.edu.ph