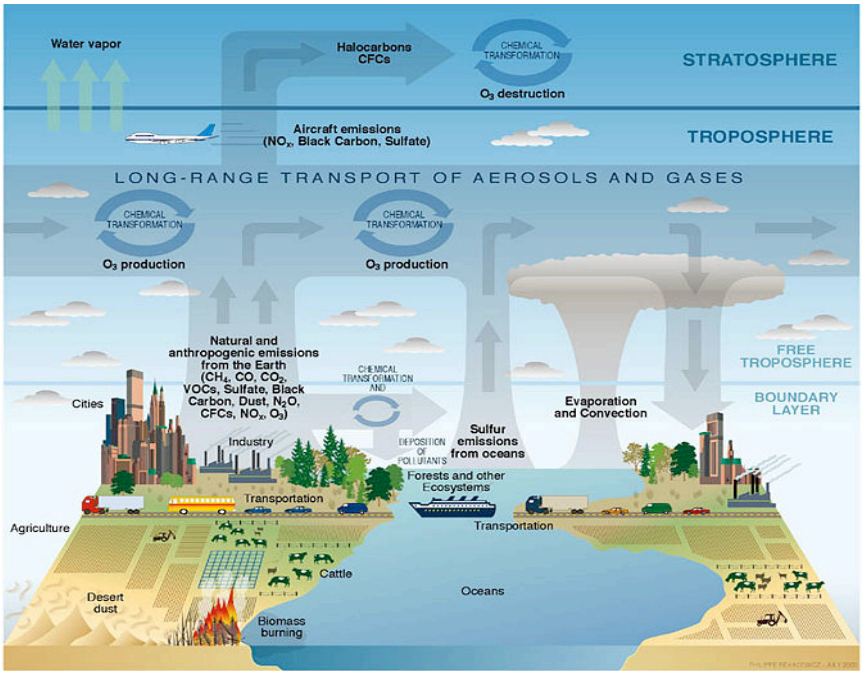
CPS 220 – Theory of Computation

REGULAR LANGUAGES

Introduction

Model (def) – a miniature representation of a thing; sometimes a facsimile

- Iraq village mockup for the Marines
- Scientific modelling - the process of generating abstract, conceptual, graphical and or mathematical models. Science offers a growing collection of methods, techniques and theory about all kinds of specialized scientific modelling (Wikipedia.org)



Reason for modeling computers

- Actual computers are very complex

Computational Model

- Accurate in some cases & not in others

  Therefore requires many different models

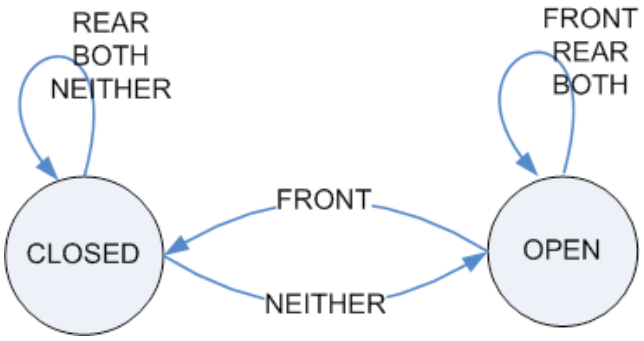
Finite Automata

Textbook Example

front and rear of an automatic door pad

front | rear

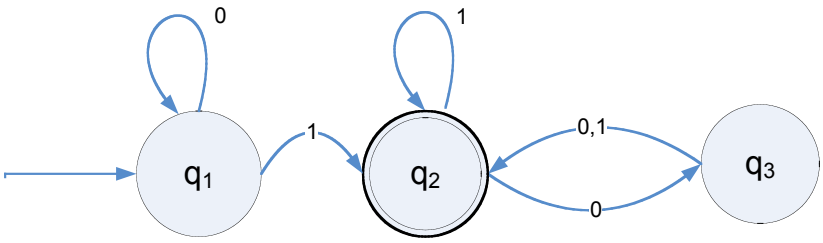
state diagram



state transition table

Input signal					
state		neither	front	rear	both
	closed	closed	open	closed	closed
	open	closed	open	open	open

Generic machine

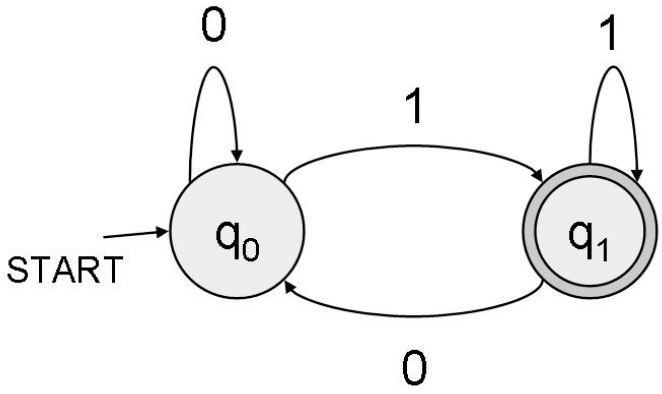


follow the progress through a string and then when all string characters are processed - either accept or reject

What does this machine do? Accept must end with a one or even number of zeros following the last 1

Example:

Transition diagram for finite automaton M:



M has two states: q_0, q_1.
q_1 is an *accepting* state.
All strings ending in 1 are accepted. That is, all odd binary numbers are accepted.

The *language* this automaton recognizes is:

L(M) = { 1, 01, 11, 001, 011, 101, 111, … } = all odd binary numbers.

String & Languages

1 Alphabet: A finite set of symbols.
Examples:

$\Sigma = \{ 0, 1 \}$
$\Sigma = \{ 0,1, 2, 3, 4, 5, 6, 7, 8, 9 \}$

2 String: A finite sequence of symbols from some alphabet.

Examples: 0100101 1345

3 Language: A set of strings chosen from some alphabet.

Examples:
L = all English words.
therefore, dog $\in$ L and DoGcaT $\notin$ L
L = all binary integers ending in 1
|L| = number of string in the language

Some notation on Strings:

4 The empty string "" is a string! It is a very important one.
We will denote it by: ε

5 Σ^k is the set of strings of length k, under the alphabet Σ.
Examples: $\Sigma = \{0,1\}$
$\Sigma^0 = \varepsilon$
$\Sigma^1 = \{0,1\}$
$\Sigma^2 = \{00,01,10,11\}$
…
$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup ...$
$\Sigma_\varepsilon = \Sigma \cup \{ \varepsilon \}$

Example: $\{0,10\}^* = \{ \varepsilon \} \cup \{0,10\} \cup \{00,010,100,1010\} \cup ...$
Note: in this case the alphabet is made up of 2 symbols - 0 and 10

6 If w is a string, then w^k is the string consisting of w repeated k times: wwww…w

Definition of Finite State Machines (Finite Automata)

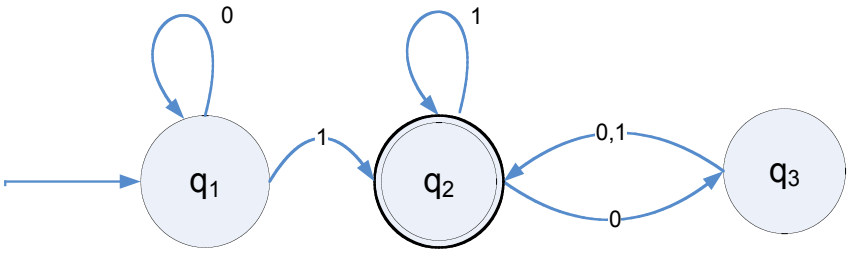
$M = (Q, \Sigma, \delta, q_0, F)$

1. Q is a *finite* set of **states**.
2. Σ is a *finite* set of symbols, the **alphabet**.
3. $\delta: Q \times \Sigma \rightarrow Q$ is the **transition function**.
4. q_0 is a **start state**.
5. F is a set of **accepting**, or **final states**.

The transition diagram is just a graphical representation.

Example:

state diagram:



$M = (Q, \Sigma, \delta, q_1, F)$
$Q = \{q1, q2, q3\}$
$\delta: Q \times \Sigma \rightarrow Q = \delta(q1,0)=q1, \delta(q1,1)=q2, \ldots$ (easier to describe with state transition table):

	0	1
*q*1	*q*1	*q*2
*q*2	*q*3	*q*2
*q*3	*q*2	*q*2

q1 is the start state
$F = \{q2\}$

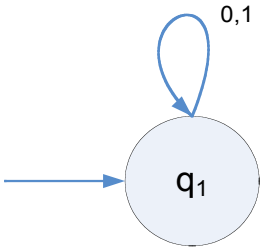
Definition of the Language of a FA

Definition of Language: We say that M accepts a language A, if A is exactly the set of strings that M accepts.

(M accepts A or M recognizes A)

L(M) = the language accepted by M.

if a machine accepts no strings - it still recognizes one language - empty language $\varnothing$
therefore consider the following finite automata M



M = (Q, Σ, δ, q_1, F)
Q = {q1,}
δ: Q × Σ → Q = δ(q1,0)=q1, δ(q1,1)=q1
q1 is the start state
F = {ε}
therefore L(M) = $\varnothing$

Formal definition of computation
Let M = (Q, Σ, δ, q_0, F) be a finite automaton, and w = $w_1w_2…w_n$ be a string. M accepts w if there exists a sequence of states r_0, r_1, r_2, …, r_n in Q, such that:

1. $r_0 = q_0$	that is, r_0 is the start state
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for i=0,…,n–1	that is, M goes from r_i to r_{i+1} on input w_{i+1}
3. $r_n \in F$	that is, r_n is an accepting state

Definition: L(A) = { w | δ^*(q_0 , w) ∈ F } (δ^*( ) simply means continued processing on string until all symbols are processed)

The Regular operations

Definition. A language is called **regular** if it is recognized by some finite automaton.

Let A and B be two languages. We can define the regular operations union, concatenation, and star, as follows:

- Union: $A \cup B = \{ x \mid x \in A \text{ or } x \in B \}$
- Concatenation: $AB = \{ xy \mid x \in A \text{ and } y \in B \}$
- Star: A^* $= \{ x_1x_2\ldots x_k \mid k \geq 0, \text{ and each } x_i \in A \}$

For example, if A = { good, bad } and B = { boy, girl }

$A \cup B$ = {good,bad,boy,girl}
AB = {goodboy, goodgirl, badboy, badgirl}
A^* = {ε, good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad, ...}

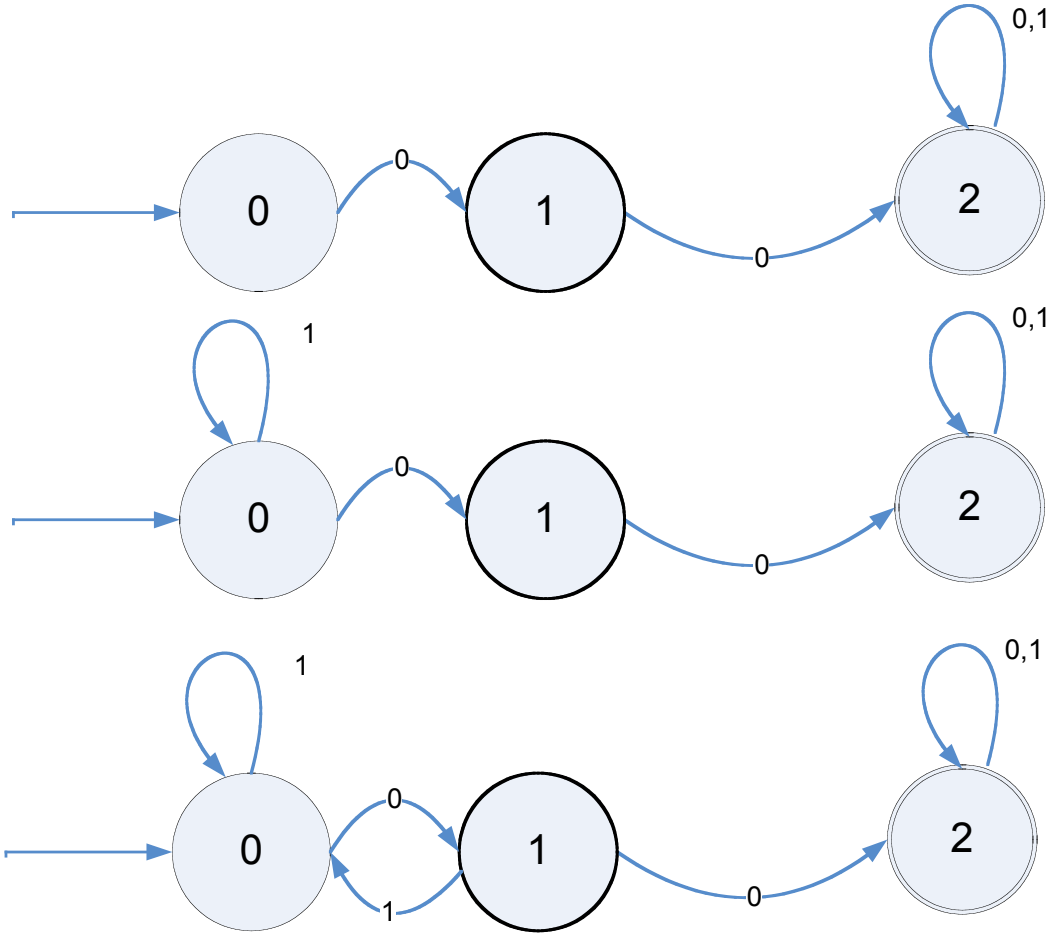
Theorem 1.25 The class of regular languages is closed under the union operation

Proof by construction. There is M1 and M2 - both machines which recognize regular languages (A1 and A2). We can construct a machine M which rcognizes the union of M1 and M2 - since a finite automaton recognizes it then it is regular. Therefore closed under the union operation. (Proof in book - page 46)

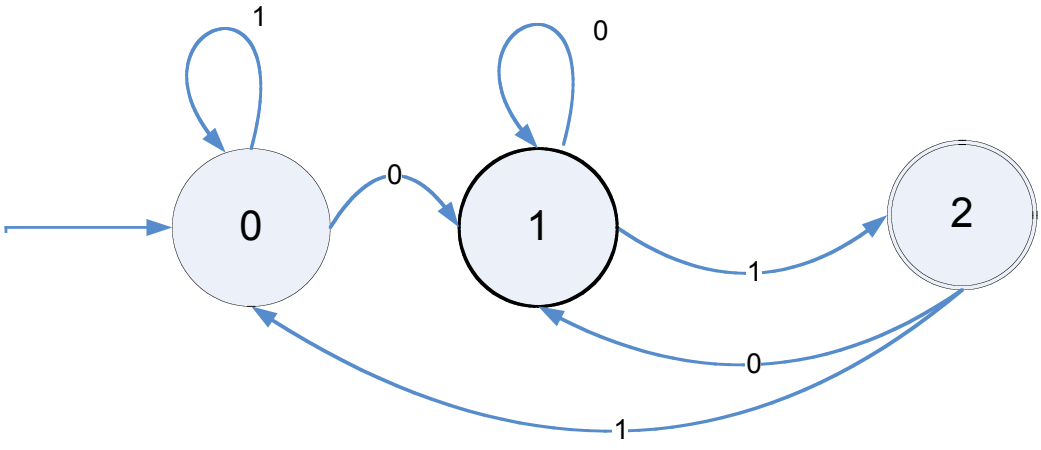
Example of a union construction. The set of all binary strings having a substring 00 or ending with 01.

First get two machine M1 and M2. M1 - all binary strings having a substring 00 and M2 - set of all binary strings ending with 01.

Construct M1: all binary strings having a substring 00 (This is called the checker method: first you build a checker then you progressively add on to it)



Construct M2: set of all binary strings ending with 01



Construct M1 ∪ M2:

1. Set of states: (Note: the book's proof p.46 uses as the states - the set of Cartesian

product of sets Q1 and Q2 - however a number of these new states will not be used.)
2. Alphabet: alphabet of both M1 and M2.
3. Transition function: $\delta(r1,r2),a) = (\delta1(r1,a) ,\delta2(r2,a))$ (see my example)
4. q0 is the pair (q1,q2)
5. F is the set of pairs which either member is an accept state for M1 and M2.

Construct a state transition table:

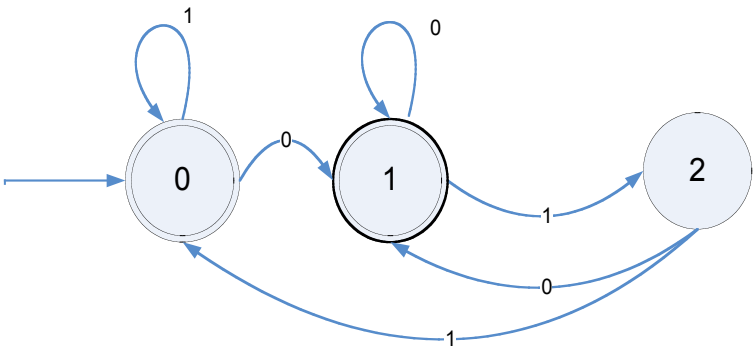
	0	1
0 0	1 1	0 0
1 1	2 1	0 2
2 2*	2 1	2 0
2 1*	2 1	2 2
2 0*	2 1	2 1
0 2*	1 1	0 0

What would be the states, starting pair, and F? What would the state diagram look like?

*Note: This construction technique also works for intersection (simply choose the F as the set of pairs which both members is an accept state for M1 and M2).

*Also we can construct the complement of a machine. How? Simply flip the final states with the states which are not final.

Example of complementation: (All strings not ending with 01)



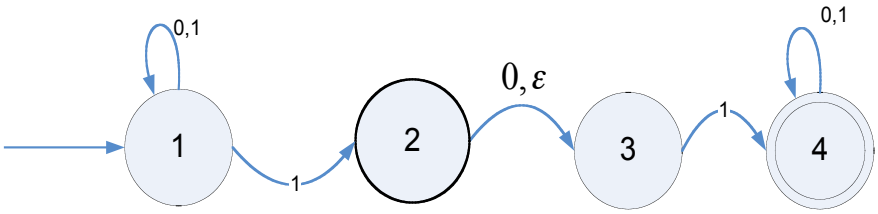
Theorem 1.26 The class of regular languages is closed under the concatenation operation however can't a

concatenation machine can not be constructed like we did with union - therefore we must introduce
a new machine nondeterministic FA.

Definition of a Nondeterministic FA

Determinism - we know what the next state will be - it is determined.
Nondeterminism - several choices may exist for the next state at any point (generalization of determinism)

Example: (all strings that contain either 101 or 11 as a substring)



What are the differences between DFA and NFA?
A NFA state may have zero, one, or many exiting arrows for each alphabet symbol
A NFA state may have arrows with alphabet or ε.

How does NFA function?
At each state - the machine splits itself into multiple execution streams - heading down each possible path.
If an execution path ends in an accept state then the string is recognized by the machine.
If a path gets stuck (can't proceed) then it simply dies away.

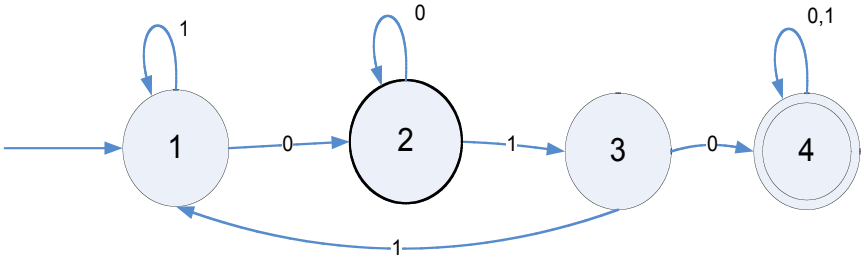
DETERMINISM NONDETERMINISM

[see figure 1.28 on page 49]

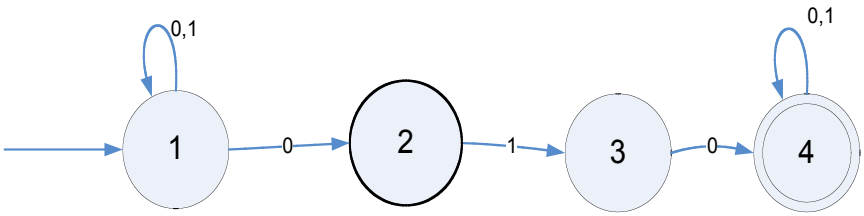
Think of nondeterminism as a *parallel search*.

Often easiler to design NFA than DFA. An example: (having substring 010)

DFA:



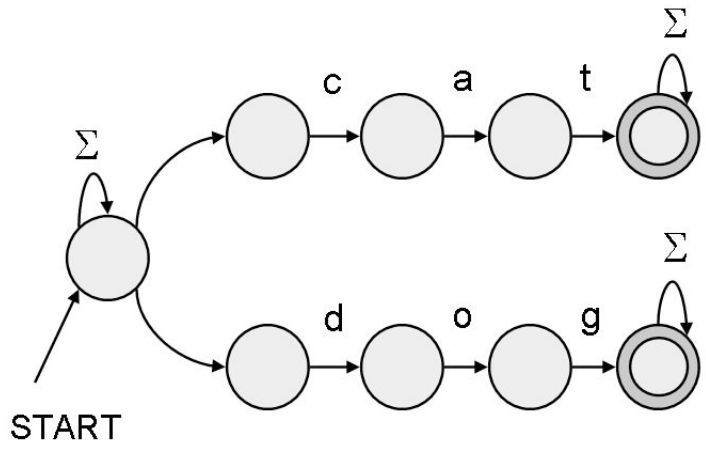
NFA:


0,1
1
0
2
1
3
0
0,1
4

ε-Transitions

An NFA can have transitions between two states that use no input symbols.

Example: (any string with substring either cat or dog)



This is just a convenience. It may reduce the number of states required, and make the machine easier to construct.
But as we will see it does not make the NFAs more "powerful" i.e. Both FA and NFA still recognize the same set of languages.

δ function for ε-NFAs

Same as a DFA except for δ:

$$\delta: Q \times \Sigma_\varepsilon \rightarrow 2^Q$$

δ takes a state in Q and a symbol in Σ and returns a subset of Q.

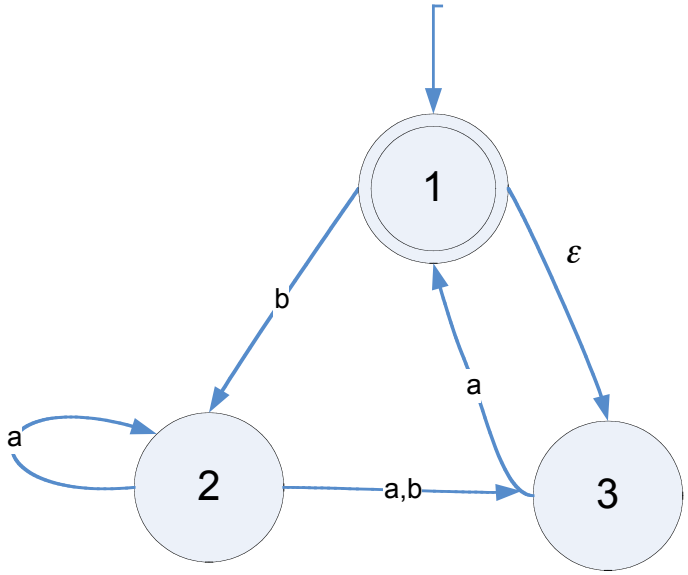


figure 1.36 from textbook (What strings does it recognize?)

state transition table

	ε	a	b
1	{3}	∅	{2}
2	∅	{2,3}	{3}
3	∅	{1}	∅

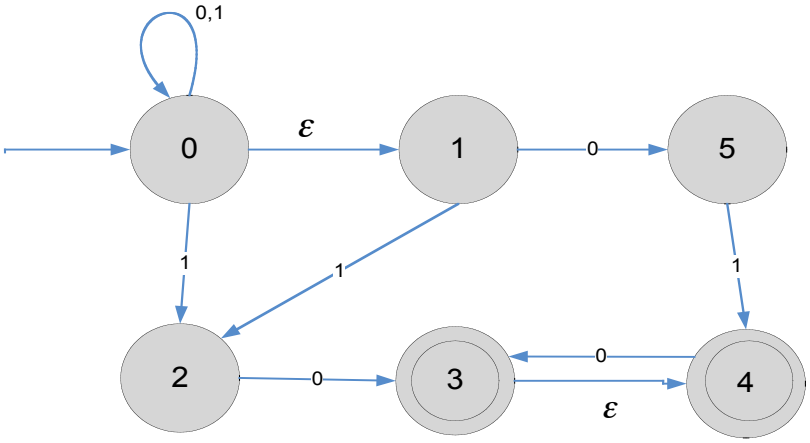
Note: the transition function now produces a set of states

Equivalence of DFAs and NFAs

NFAs are convenient, and they may look much more “powerful” than DFAs,
but in fact anything that an NFA can do, some DFA can do as well - therefore they are equivalent.

Theorem. A language A is accepted by an NFA, if and only if it is accepted by a DFA.
Proof. Instead of carefully walking out way through the proof given in the book. We will look at 2 examples of converting NFA to DFAs. (Proof by construction)

NFA



Defiintion of ε–closure - The ε–closure of a subset $A \subseteq Q$ is the set of states that can be reached
from a state q in A by ε–moves (including the move from q to q)

For example (above NFA) -
ε–closure ({q0})= {q0,q1}, ε–closure ({q3})= {q3,q4}, ε–closure ({q5})= {q5}, ...

What would be ε–closure ({q0}) if there was a ε–move from q1 to q5. ε–closure ({q0})= {q0,q1, q5}.

NFA-->DFA

Step 1. Let $Q_\varepsilon = \varepsilon - closure(\{q0\})$ as the inital state, and let F' = ∅ be the set of final states. Let Q'= $\{Q_\varepsilon\}$. If $Q_\varepsilon \cap F \neq \varnothing$, then add

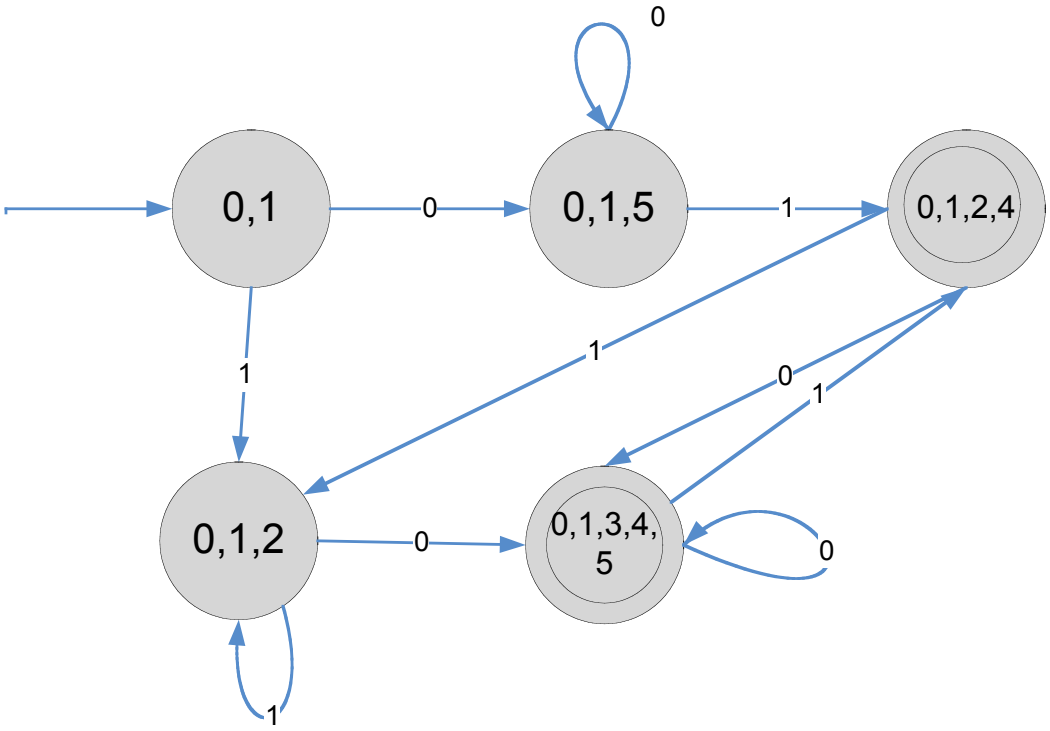
Step 2. Repeat the following until $\delta'(Q_x, a)$ is defined for all $Q_x \in Q'$ and all a ∈{0,1} :

(1) Select $Q_x \in Q'$ and a ∈{0,1} such that $\delta(Q_x, a)$ is not yet defined.

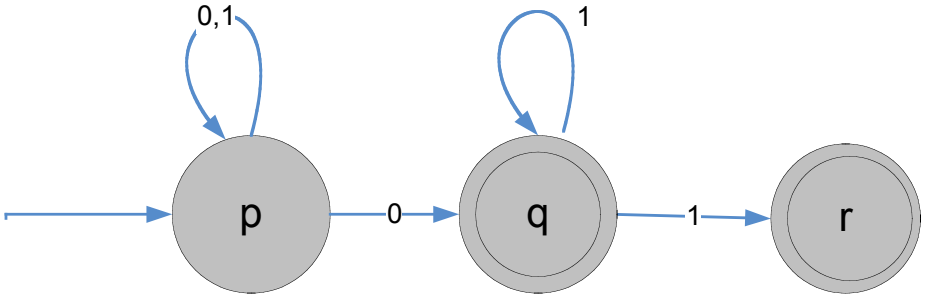
(2) Let $Q_{xa} = \delta(Q_x, a)$

(3) If $Q_{xa} \notin Q'$, then add Q_{xa} to Q', and also add it to F' if $Q_{xa} \cap F \neq \varnothing$.

δ'	0	1
$Q_\varepsilon = \{q0, q1\}$	$\{q0, q1, q5\}$	$\{q0, q1, q2\}$
$Q_0 = \{q0, q1, q5\}$	$\{q0, q1, q5\} = Q_0$	$\{q0, q1, q2, q4\}$
$Q_1 = \{q0, q1, q2\}$	$\{q0, q1, q3, q4, q5\}$	$\{q0, q1, q2\} = Q_1$
$Q_{01} = \{q0, q1, q2, q4\}$	$\{q0, q1, q3, q4, q5\}$	$\{q0, q1, q2\} = Q_1$
$Q_{10} = \{q0, q1, q3, q4, q5\}$	$\{q0, q1, q3, q4, q5\} = Q_{10}$	$\{q0, q1, q2, q4\} = Q_{01}$



Convert this NFA to a DFA:



Corollary 1.40 - A language is regular iff some NFA recognizes it.

Closure properties of regular languages

Definition. A collection of objects is **closed** under some operation if applying that operation to members of the collection returns an object still in the collection.

Example:

Integers, Z = { …, -3, -2, -1, 0, 1, 2, 3, ….. }, are closed under +

$-3 + 5 = 2 \in Z$
$-3 + (-2) = -5 \in Z$

Formally, $n, m \in Z$ implies that $n + m \in Z$

However, integers are not closed under /

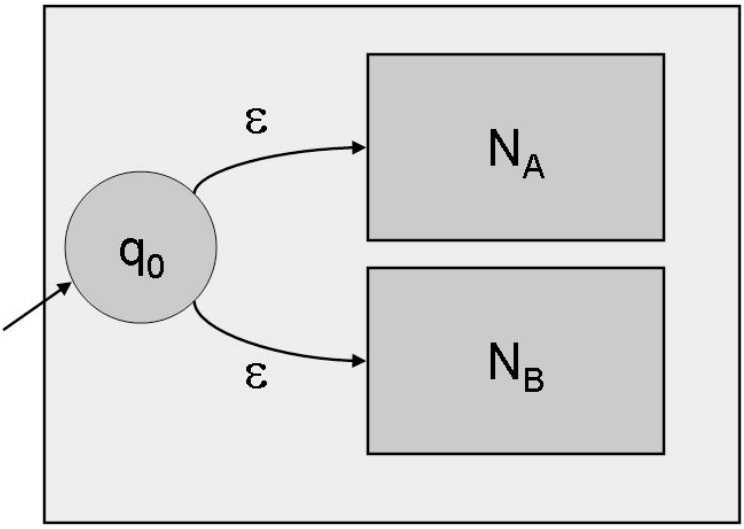
$2 / 4 = 0.5 \notin Z$

Theorem. The class of regular languages is closed under union.
(Already proven when we constructed a DFA which was the union of 2 DFAs)

Proof. We need to show that for any regular languages A and B, $A \cup B$ is regular.

Let A be regular. Then there is an NFA N_A accepting A.
Let B be regular. Then there is an NFA N_B accepting B.

Here is an NFA N accepting $A \cup B$:

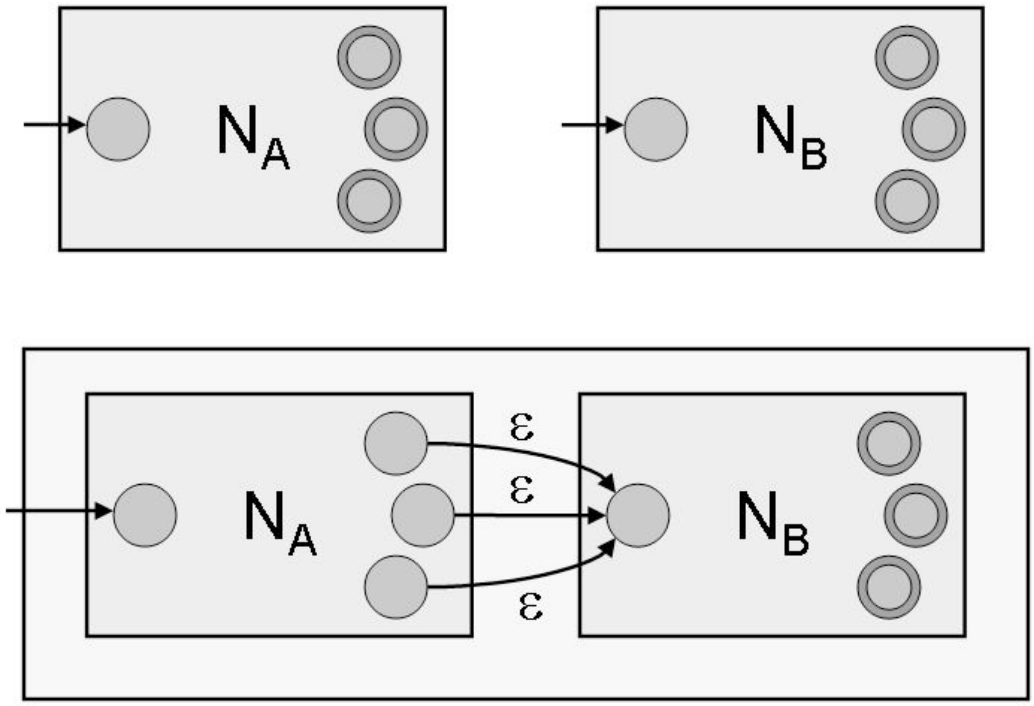


Theorem. The class of regular languages is closed under concatenation.

Proof. We need to show that for any regular languages A and B, then AB is regular.

Let A be regular. Then there is an NFA N_A accepting A.
Let B be regular. Then there is an NFA N_B accepting B.

Here is an NFA N accepting AB:



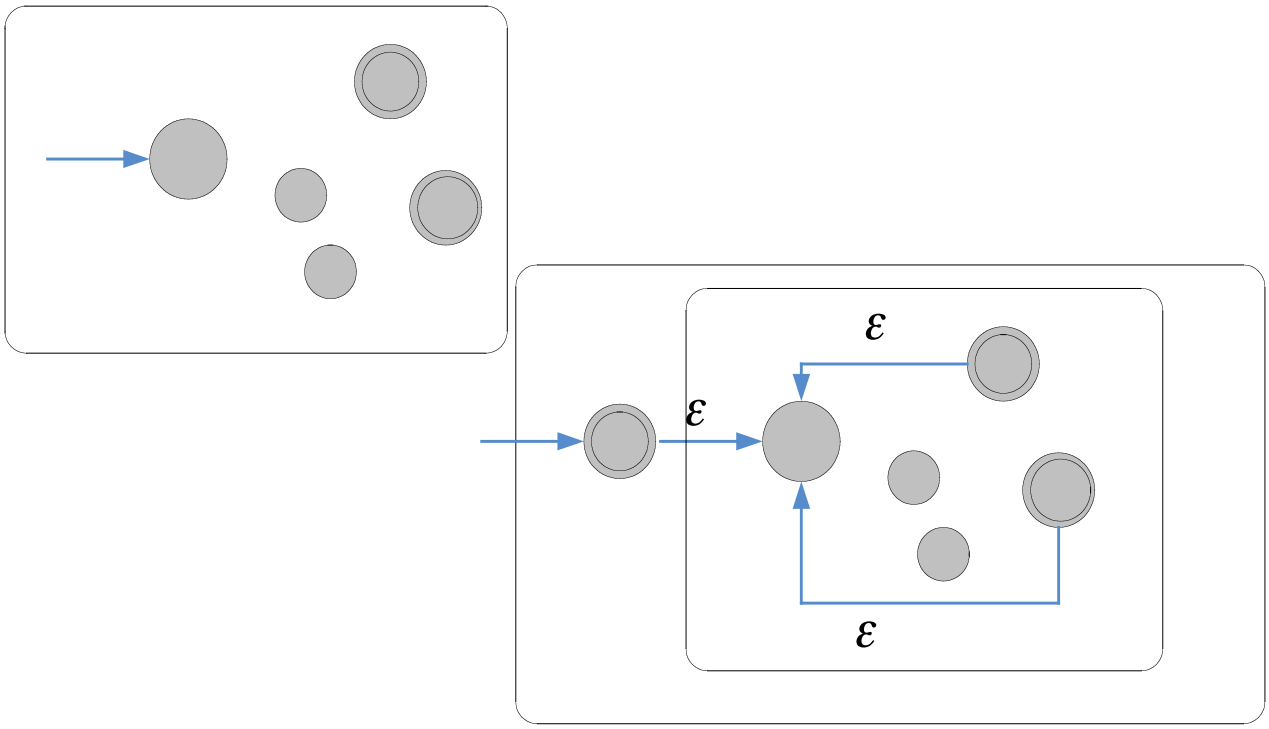
Theorem. The class of regular languages is closed under star. (Kleene closure)

Proof. We need to show that for any regular language A, A^* is also regular.

Let A be regular. Then there is an NFA N_A accepting A.

Here is an NFA N accepting A^*:

N



N'=N*

References:

Introduction to the Theory of Computation (2nd ed.) Michael Sipser
Problem Solving in Automata, Languages, and Complexity Ding-Zhu Du and Ker-I Ko