Formal Definition of DFA

Described by a 5-tuple:

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

Q = Set of States

<u>Finite</u> Number of states

Σ = Alphabet, a <u>Finite</u> Set of Symbols

<u>by default, function means "total" function!</u>

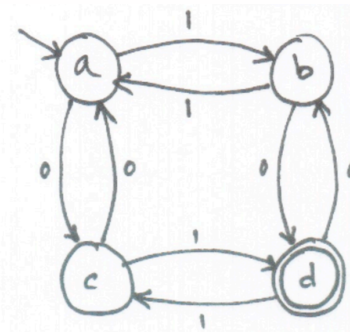
δ = The TRANSITION FUNCTION

$$\delta : Q \times \Sigma \rightarrow Q$$

q_0 = The STARTING STATE

$q_0 \in Q$ (or "INITIAL" STATE)

F = The set of ACCEPT states (or "FINAL" STATES) $F \subseteq Q$



$Q = \{a, b, c, d\}$

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

$q_0 = a$

$F = \{d\}$

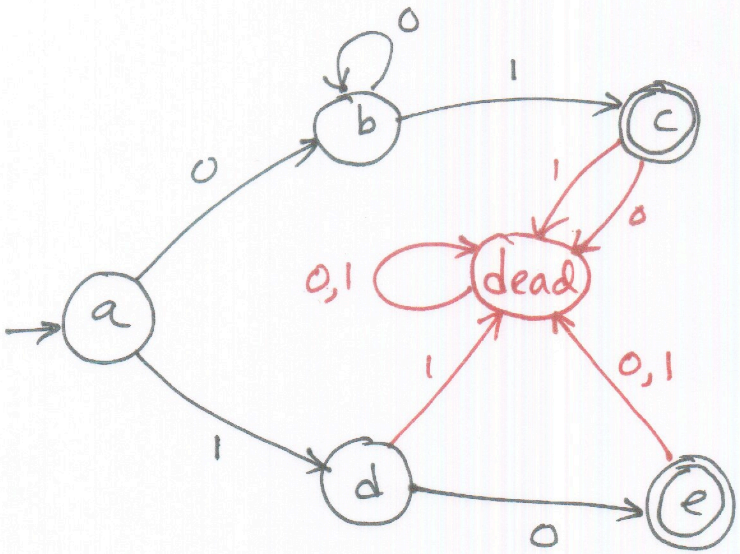
$\delta =$

Symbols

States

	0	1
a	c	b
b	d	a
c	a	d
d	b	c

Alternative Definition of Transition



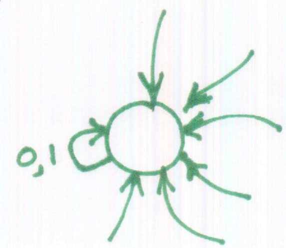
δ is a function (total)

FORMALLY, must be defined

$\delta(c, 1) = ?$

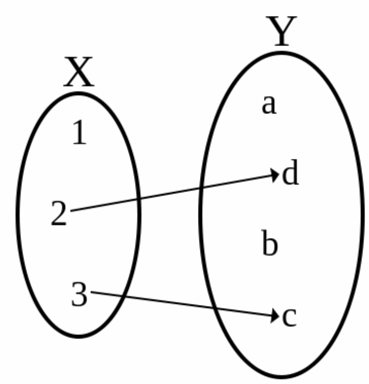
If some transitions are missing, add a dead state.

(Often, prefer not to show dead state.)

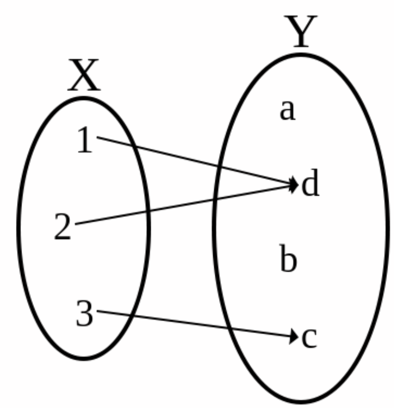


if we change the transition function δ from a total function to a *partial* function then we don't need to include trap state because whenever $\delta(s, a)$ is undefined, it goes to the trap state.

note that the definition of computation remains unchanged.



partial



total

https://en.wikipedia.org/wiki/Partial_function

Formal Definition of Computation

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

Let $w = w_1 w_2 \cdots w_N$ be a string
where $w_i \in \Sigma$

M accepts w iff. there is a
sequence of states
$r_0, r_1, r_2, \cdots, r_N$ in Q
such that
$r_0 = q_0$
$\delta(r_i, w_{i+1}) = r_{i+1}$ for $0 \leq i < N$
$r_N \in F$

We say...

M "recognizes" Language A
if $A = \{ w \mid M \text{ accepts } w \}$

Redefine computation: Extend δ to δ^*

Defining the computation of an FA $M=(Q,\Sigma,q_0,A,\delta)$.

$\delta(q,a) =$ which state to go to from q after reading a

$\delta^*(q,s) = $ which state we end in after starting in q and reading the string s

e.g. aabba

Extended transition function $\delta^* : Q \times \Sigma^* \to Q$:

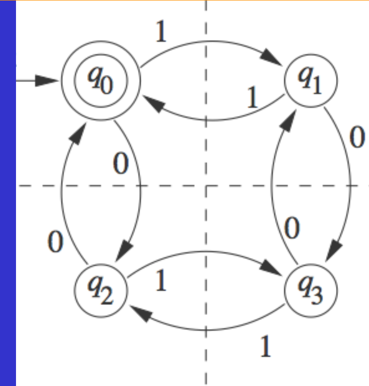
a string

1) For every $q \in Q$, let $\delta^*(q,\Lambda) = q$

2) For every $q \in Q$, $y \in \Sigma^*$, and $\sigma \in \Sigma$, let $\delta^*(q,y\sigma)$ $= \delta(\underbrace{\delta^*(q,y)}_{r},\sigma)$

$q \xrightarrow{y_1} \circ \xrightarrow{y_2} \circ \cdots \xrightarrow{y_{|y|}} r \xrightarrow{\sigma} *$

We say that a string $x \in \Sigma^*$ is **accepted by M** iff $\delta^*(q_0,x) \in A$.



$\delta^*(q_0, 10) = q_3$
$\delta^*(q_3, 111) = q_2$
$\delta^*(q_2, \varepsilon) = q_2$
$\delta^*(q_1, 1) = q_0$

$$\delta^*(q,w) = \begin{cases} \delta(\delta^*(q,x),a) & \text{where } w = xa \text{ and } x \in \Sigma^*, a \in \Sigma \\ q & \text{where } w = \epsilon \end{cases}$$

$$\boxed{L((Q,\Sigma,\delta,q_0,F)) = \{w \in \Sigma^* \mid \delta^*(q_0,w) \in F\}}$$

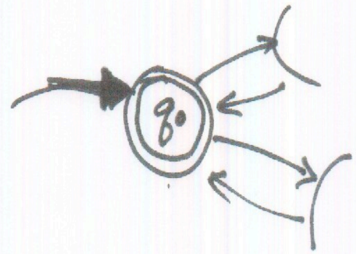
Q1: what about decomposing w = ax or even w = xy?
Q2: what about partial function δ?

δ^* is <u>*not*</u> *defined in Sipser, but is in all other textbooks. This is probably one of the biggest flaws of Sipser book.*

Language, String, Machine

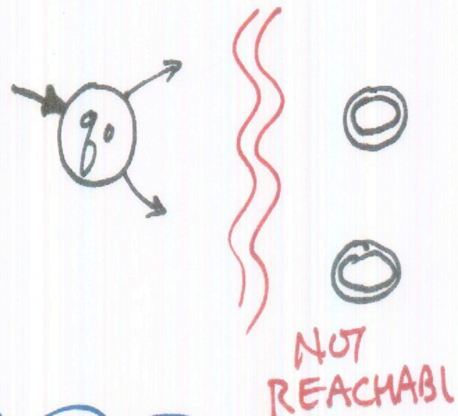
THE EMPTY STRING

ϵ (epsilon, ε)



THE EMPTY LANGUAGE

$\emptyset = \{\}$



NOTE:

$\{\epsilon\} \neq \emptyset$

$\epsilon \neq \emptyset$

If a machine accepts NO strings then it recognizes the EMPTY LANGUAGE

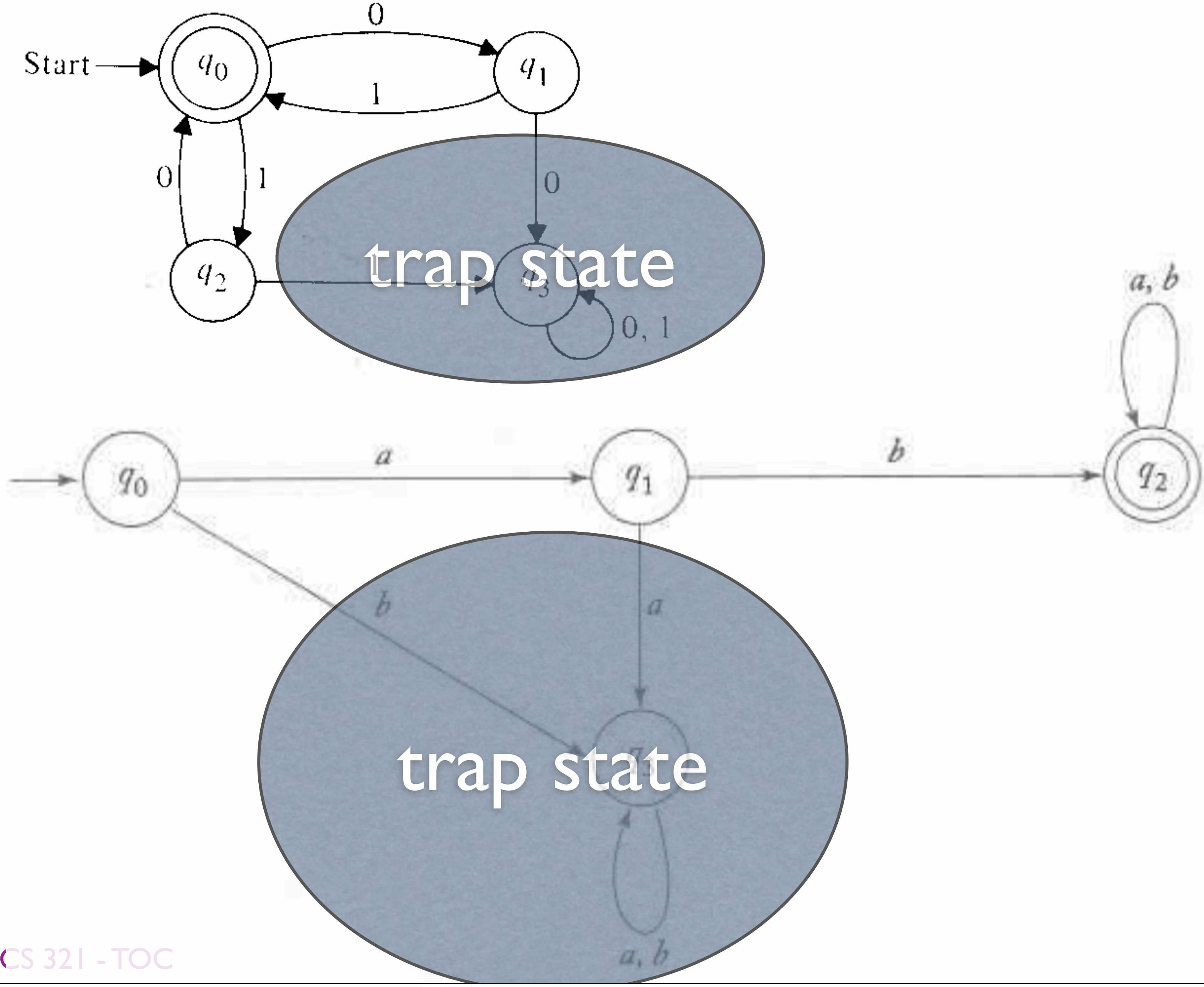
7

does M accept any string?

does M accept empty string?

does M recognizes the empty language?

What's the language of ...



Divisible by 3

Q1: language over {a,b,c} s.t. the # of a's is divisible by 3

hint: need 3 states:

state 0: (#a's) % 3 == 0

state 1: (#a's) % 3 == 1

state 2: (#a's) % 3 == 2

wait... what about b/c?

Q2: language over {a,b,c} s.t. (the # of a's) - (the # of b's) is divisible by 3

hint: still 3 states:

state 0: ((#a's) - (#b's)) % 3 == 0

state 1: ((#a's) - (#b's)) % 3 == 1

state 2: ((#a's) - (#b's)) % 3 == 2

Binary Number Divisible by 3

Binary Numbers that are divisible by 3.

$\Sigma = \{0, 1\}$ $L = \{\underset{0}{0}, \underset{3}{11}, \underset{6}{110}, \underset{9}{1001}, \underset{12}{1100}, \underset{15}{1111} \ldots\}$

As we scan a binary number, what does each bit do to the value?

$101101010110001 \begin{cases} 0 \rightarrow 2(x) \\ 1 \rightarrow 2(x)+1 \end{cases}$

3X ← Divisible by 3

3X+1, 3X+2 ← Not divisible by 3

if we see 0:

$2(3X)^{A} = 3(2X)$ A

$2(3X+1)^{B} = 3(2X)+2$ C

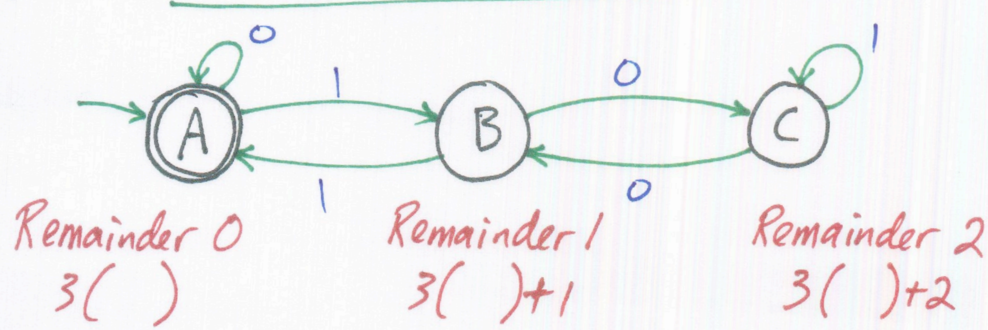
$2(3X+2)^{C} = 3(2X+1)+1$ B

if we see 1:

$2(3X^{A})+1 = 3(2X)+1$ B

$2(3X+1^{B})+1 = 3(2X+1)$ A

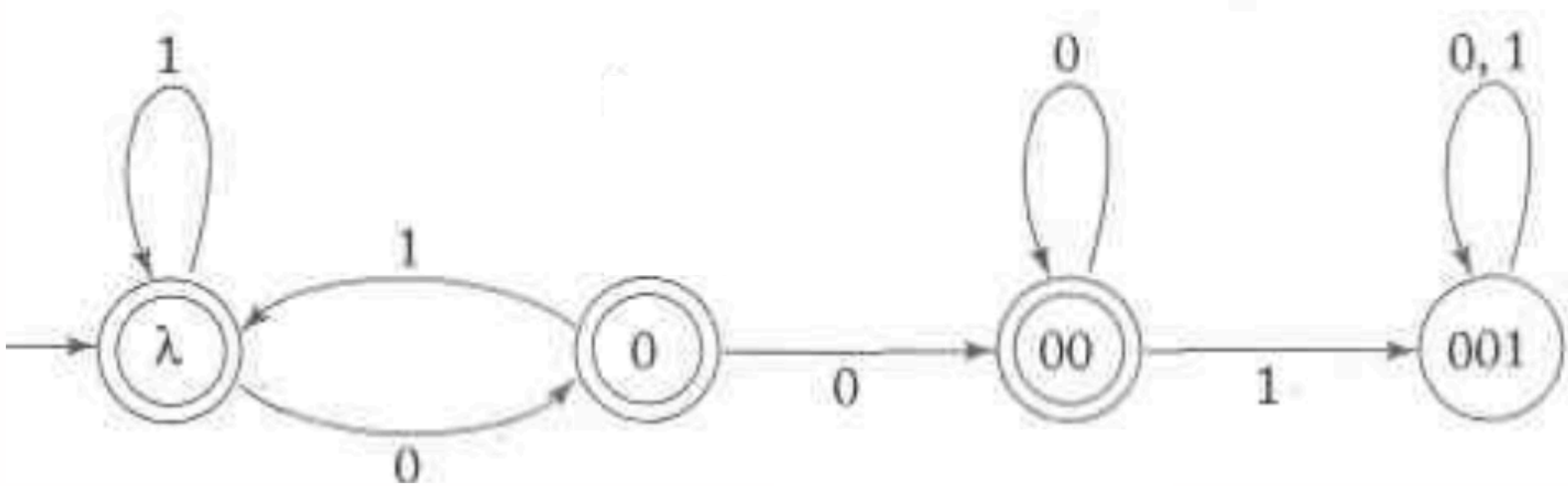
$2(3X+2^{C})+1 = 3(2X+1)+2$ C



- still just need 3 states
- decimal num divisible by 3?
  - again, 3 states
- general strategy in base *d*
  - divisible by *n* => *n* states
  - [0] [1] ... [*n*-1]
    - [*i*] = { *x* | *x* % *n* = *i* }
  - at state [*q*], on digit *i*
    - goto state [(*q* x *d* + *i*) % *n*]
- but it's possible to use *less* than *n* states!

Construct Finite Automaton for ...

- any string that does not contain 001 in it
- try simpler problem: a string that does contain 001

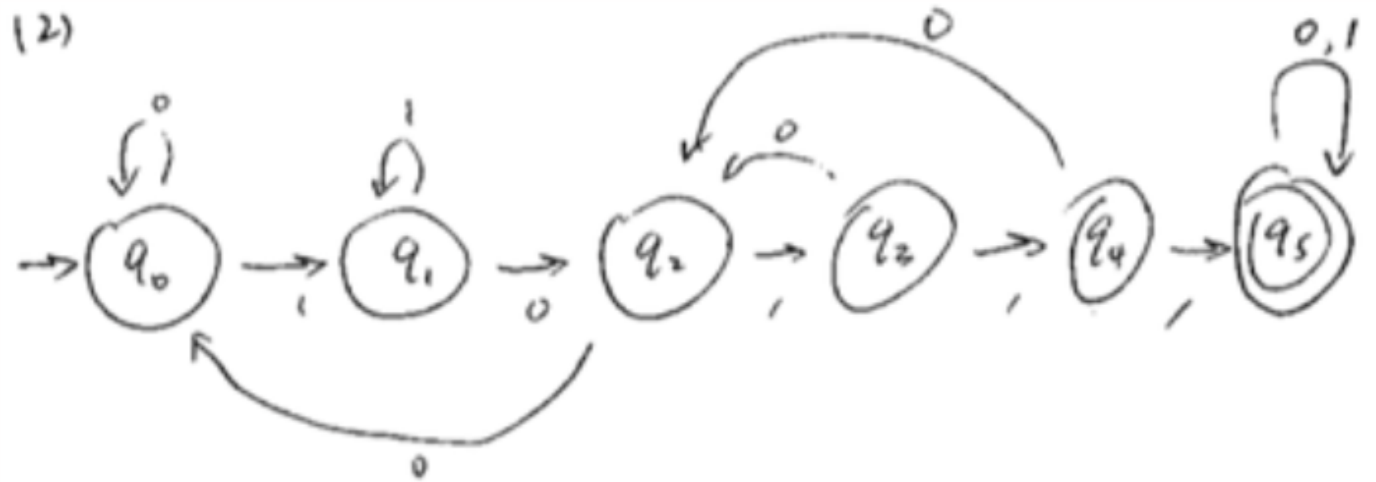


this reminds you of KMP string matching

https://www.ics.uci.edu/~eppstein/161/960222.html

Construct Finite Automaton for ...

- any string that does contain 10111 in it



general strategy for "contains pattern $a_0a_1...a_{n-1}$":

backbone: at state i on a_i goto state i+1 (n+1 states)

deviations: at state i, on any input b != a_i

go back to the rightmost such state j,

where prefix $a_0a_1...a_{j-1}$ == suffix $a_{i-j+1}...a_{i-1}b$

because we can reuse such suffix and

don't need to restart from the very beginning

https://www.ics.uci.edu/~eppstein/161/960222.html

Complement Language

Notation

$L(M_1)$ = The language that M_1 recognizes.
= The set of strings over $\{0,1\}^*$ that contain 0011 as a substring.

$L(M_2)$ = The set of strings over $\{0,1\}^*$ that do not contain 0011.

COMPLIMENTING A LANGUAGE

They are sets, after all.

$L(M_1) = \overline{L(M_2)}$

The "Universe"

All possible strings made with symbols from the alphabet.

$\Sigma = \{0,1\}$ Universe $= \{0,1\}^*$

Set compliment is always relative to some Universe; (implicitly).

$\overline{L} \triangleq \Sigma^* \setminus L = \{x \mid x \in \Sigma^*, x \notin L\}$

if $L = L(M)$ and $M = (Q, \Sigma, \delta, q, F)$ then

$\overline{L} = L(\overline{M})$ where $\overline{M} = (Q, \Sigma, \delta, q, \overline{F})$

where $\overline{F} = Q \setminus F$

just flip final and non-final!

9

Prove: Complement of Regular is Regular

Proof: For every regular language L, by definition of regular language, there must be a DFA M s.t. $L(M) = L$. let $M = (Q, \Sigma, \delta, q_0, F)$ where δ is a **total** function, let us construct another DFA $\overline{M} = (Q, \Sigma, \delta, q_0, \overline{F})$ where $\overline{F} = Q \backslash F$.

For every string $w \in L$, it will end up in a state $q \in F$ in M, and it will end up in the same state in $\overline{M}$ which rejects w since $q \notin \overline{F}$; similarly, for every string w' in the complement language, i.e., $w \in \Sigma^* \setminus F$, it will end up in a state $q' \notin F$ in M, and it will end up in the same state in $\overline{M}$ which accepts w since $q' \in \overline{F}$. So $\overline{M}$ accepts all strings in $\Sigma^* \setminus L$ **and only those**, which means the complement language $\Sigma^* \setminus L$ is recognized by DFA $\overline{M}$, thus regular. $\square$

Note, however, that if δ is a partial function (i.e., trap state omitted), this proof does not work (why?). You would have to add a trap state and all trap transitions to make δ a total function first.

Rewrite/Simplify using δ^* notation

Proof: For every regular language L, by the definition of regular language, there must be a DFA M s.t. $L(M) = L$. let $M = (Q, \Sigma, \delta, q_0, F)$ where δ is a **total** function, we construct another DFA $\overline{M} = (Q, \Sigma, \delta, q_0, \overline{F})$ where $\overline{F} = Q \backslash F$.

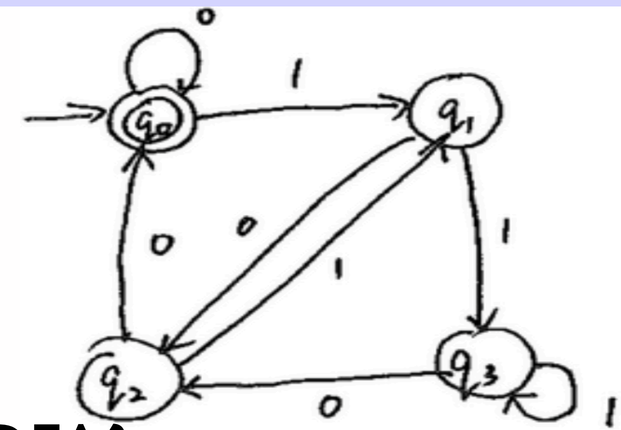
For every string $w \in L$, there exists $q \in Q$ s.t. $\delta^*(q_0, w) = q \in F$ in M, and $\delta^*(q_0, w) = q \notin \overline{F}$ in $\overline{M}$ which rejects w; similarly, for every string w' in the complement language, i.e., $w' \in \Sigma^* \setminus F$, there exists $q' \in Q$ s.t. $\delta^*(q_0, w') = q' \notin F$ in M, and $\delta^*(q_0, w') = q' \in \overline{F}$ in $\overline{M}$ which accepts w. So $\overline{M}$ accepts all strings in $\Sigma^* \setminus L$ and **nothing else**, which means the complement language $\Sigma^* \setminus L$ is recognized by DFA $\overline{M}$, thus regular. $\square$

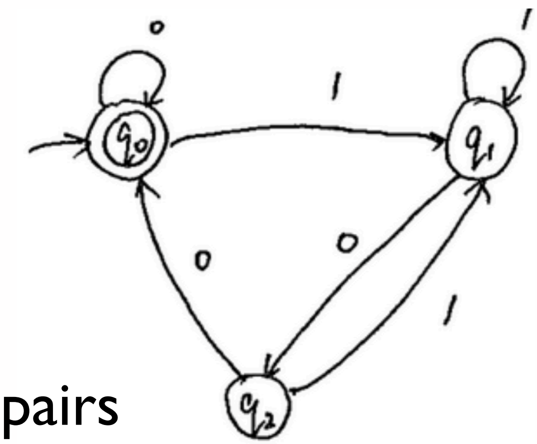
$$\boxed{\delta^*(q, w) = \begin{cases} \delta(\delta^*(q, x), a) & \text{where } w = xa \text{ and } x \in \Sigma^*, a \in \Sigma \\ q & \text{where } w = \epsilon \end{cases}}$$

$$\boxed{L((Q, \Sigma, \delta, q_0, F)) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}}$$

Binary Number Divisible by 4

- 4-state solution (trivial)
- 3-state solution (merge q1 w/ q3)
- in general, how do you:
  - reduce a DFA to a smaller but equivalent DFA?
    - see Linz 2.4 or Sipser problem 7.42 (p. 327)
    - will discuss later after NFA
  - test if two DFAs are equivalent?
    - follow pairs of states, check if all visited state-pairs agree on finality (both accept or both reject)
    - $O(n^2\ \Sigma)$ time and space

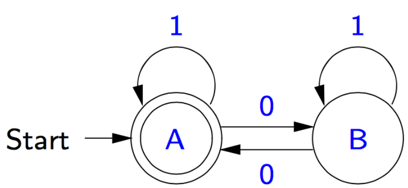


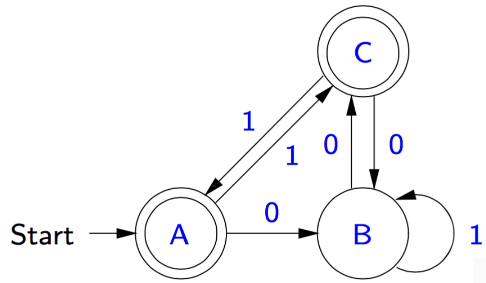


https://www.cse.iitb.ac.in/~trivedi/courses/cs208-spring14/lec05.pdf

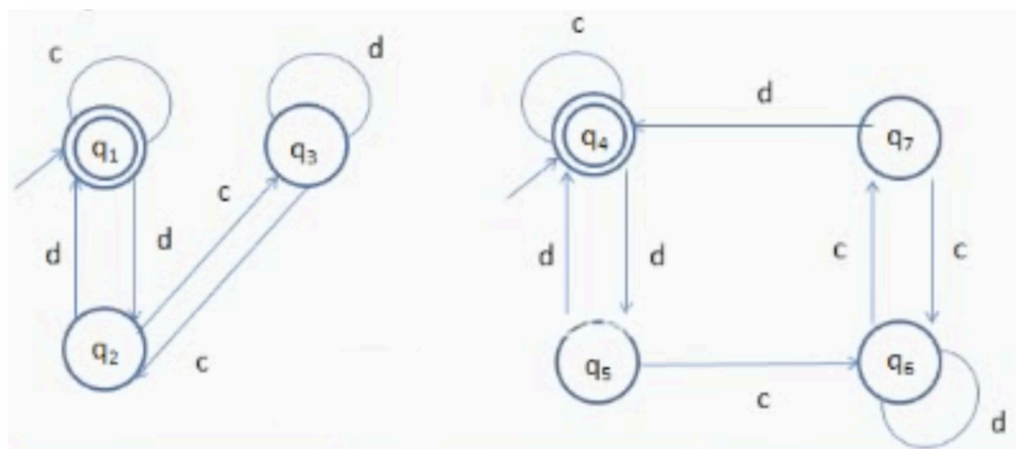
Test if two DFAs are equivalent

- traverse all state-pairs and make sure each pair agrees on “finality” (both accept or both reject)





pair	final?	on 0	on 1
(A, A)	(y, y)	(B, B)	(A, C)
(B, B)	(n, n)	(A, C)	(B, B)
(A, C)	(y, y)	(B, B)	(A, A)
no new pairs found			



Q Q'	Q_c Q_c'	Q_d Q_d'
q_1 , q_4	q_1 , q_4	q_2 , q_5
q_2 , q_5	q_3 , q_6	q_1 , q_4
q_3 , q_6	q_2 , q_7	q_3 , q_6
q_2 , q_7	q_3 , q_6	q_1 , q_4

Proof by Induction (Linz 1.1-1.2, Sipser 0.4)

- Theorem to prove: $|uv| = |u| + |v|$
- first define string length rigorously and inductively:
  - $\cancel{|a| = 1}$, $|\varepsilon| = 0$, $|wa| = |w| + 1$
- now prove the Theorem by induction on $|v|$
  - base case: $|v| = 0$, then $v = \varepsilon$, so $|uv|=|u|=|u|+0=|u|+|v|$
  - inductive case: assume Theorem holds for any $|v|$ of length $0 \ldots n$
    Now take any v of length $n+1$. Let $v = wa$, then $|v| = |w|+1$ (by def.)
    - then $|uv| = |uwa| = |uw|+1$ (by definition)
    - by induction hypothesis (applicable to any w of length n)
      - $|uw| = |u|+|w|$, so that $|uv| = |u|+|w|+1 = |u|+|v|$

$\boxed{\text{HW: prove by induction on } |u|}$

What's wrong with this proof?

Theorem(?!): All horses are the same color.

Proof: Let $P(n)$ be the predicate "in all non-empty collections of n horses, all the horses are the same color." We show that $P(n)$ holds for all n by induction on n (using 1 as the base case).

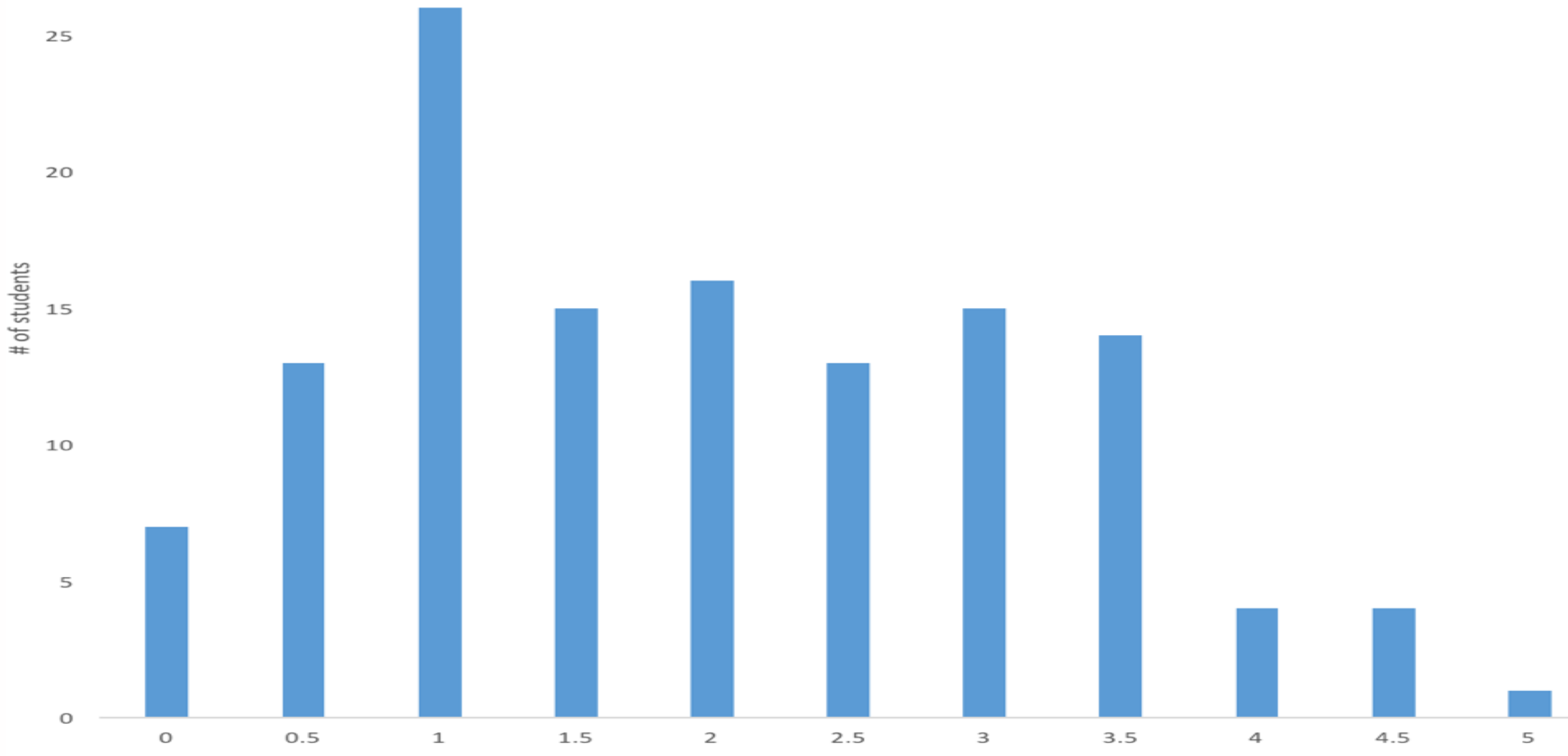
Base case: Clearly, $P(1)$ holds.

Induction case: Given $P(n)$, we must show $P(n+1)$.

Consider an arbitrary collection of $n+1$ horses. Remove one horse temporarily. Now we have n horses and hence, by the induction hypothesis, these n horses are all the same color. Now call the exiled horse back and send a different horse away. Again, we have a collection of n horses, which, by the induction hypothesis, are all the same color. Moreover, these n horses are the same color as the first collection. Thus, the horse we brought back was the same color as the second horse we sent away, and all the $n+1$ horses are the same color.

Quiz 1 scores and Projected Final Grade

- mean: 2.0, median: 2



projected final grade	0 0.5	1	1.5	2	2.5	3	3.5	4 4.5 5
	F	C/C-	C+	B-	B	B+	A-	A

Regular Language

DEFINITION

A language is a **REGULAR LANGUAGE** iff some Finite State Machine recognizes it.

What languages are NOT regular?

Anything that requires memory.

The F.S.M. memory is <u>very</u> limited

Cannot store the string.

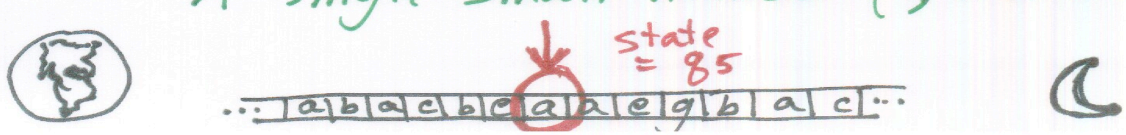
Cannot "count."

Not Regular:

ww $\qquad \underbrace{01101}_{w}\underbrace{01101}_{w}$

$0^N 1^N$ $\qquad \underbrace{000000}_{6}\underbrace{111111}_{6}$

Imagine a String from here to the Moon. You are trying to recognize it. Your only memory: A single small number (i, # of states



Regular Operations

UNION

$$A \cup B = \{x \mid x \in A \text{ or } x \in B\}$$

CONCATENATION

$$A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$$

STAR "closure"

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

EXAMPLE

$$\Sigma = \{a, b, c, \ldots z\}$$
$$A = \{aa, b\}$$
$$B = \{x, yy\}$$
$$A \cup B = \{aa, b, x, yy\}$$
$$A \circ B = \{aax, aayy, bx, byy\}$$
$$A^* = \{\epsilon, aa, b, aaaa, aab, baa, bb, aaaaaa, aaaab, aabaa, aabb, \ldots$$

- other operations:
  - intersection
  - complement
  - difference
- regular languages are <u>closed</u> under all these operations