Deterministic Finite Automata

Languages

- A *language* is a subset of Σ* for some alphabet Σ.
- Example: The set of strings of 0's and 1's with no two consecutive 1's.
- L = {ϵ, 0, 1, 00, 01, 10, 000, 001, 010, 100, 101, 0000, 0001, 0010, 0100, 0101, 1000, 1001, 1010, . . . }

Hmm… 1 of length 0, 2 of length 1, 3, of length 2, 5 of length 3, 8 of length 4. I wonder how many of length 5?

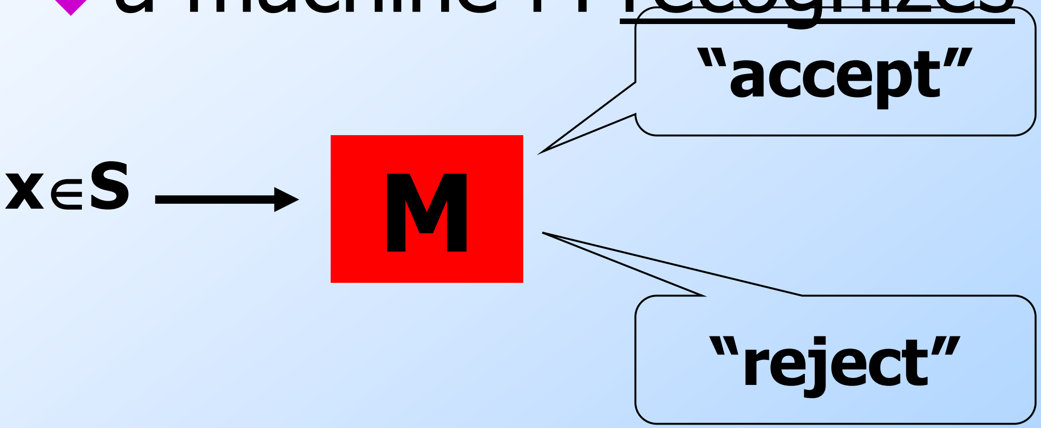
Languages

- A *language* is a subset of Σ* for some alphabet Σ.
- Example: The set of strings of 0's and 1's with no two consecutive 1's.
- L = {ϵ, 0, 1, 00, 01, 10, 000, 001, 010, 100, 101, 0000, 0001, 0010, 0100, 0101, 1000, 1001, 1010, . . . }

Hmm… 1 of length 0, 2 of length 1, 3, of length 2, 5 of length 3, 8 of length 4. I wonder how many of length 5?

Recognizing Languages

- Let L be a language $\subseteq S$
- a machine M recognizes L if



"accept" if and only if $x \in L$

"reject" if and only if $x \notin L$

Finite Automaton

The most simple machine that is not just a finite list of words.

“Read once”, “no write” procedure.

Typical is its limited memory.
Think cell-phone, elevator door, etc.

Finite Automata

- Two types – both describe what are called **regular languages**
    - Deterministic (DFA) – There is a fixed number of states and we can only be in one state at a time
    - Nondeterministic (NFA) –There is a fixed number of states but we can be in multiple states at one time
- While NFA's are more expressive than DFA's, we will see that adding nondeterminism does not let us define any language that cannot be defined by a DFA.
- One way to think of this is we might write a program using a NFA, but then when it is "compiled" we turn the NFA into an equivalent DFA.

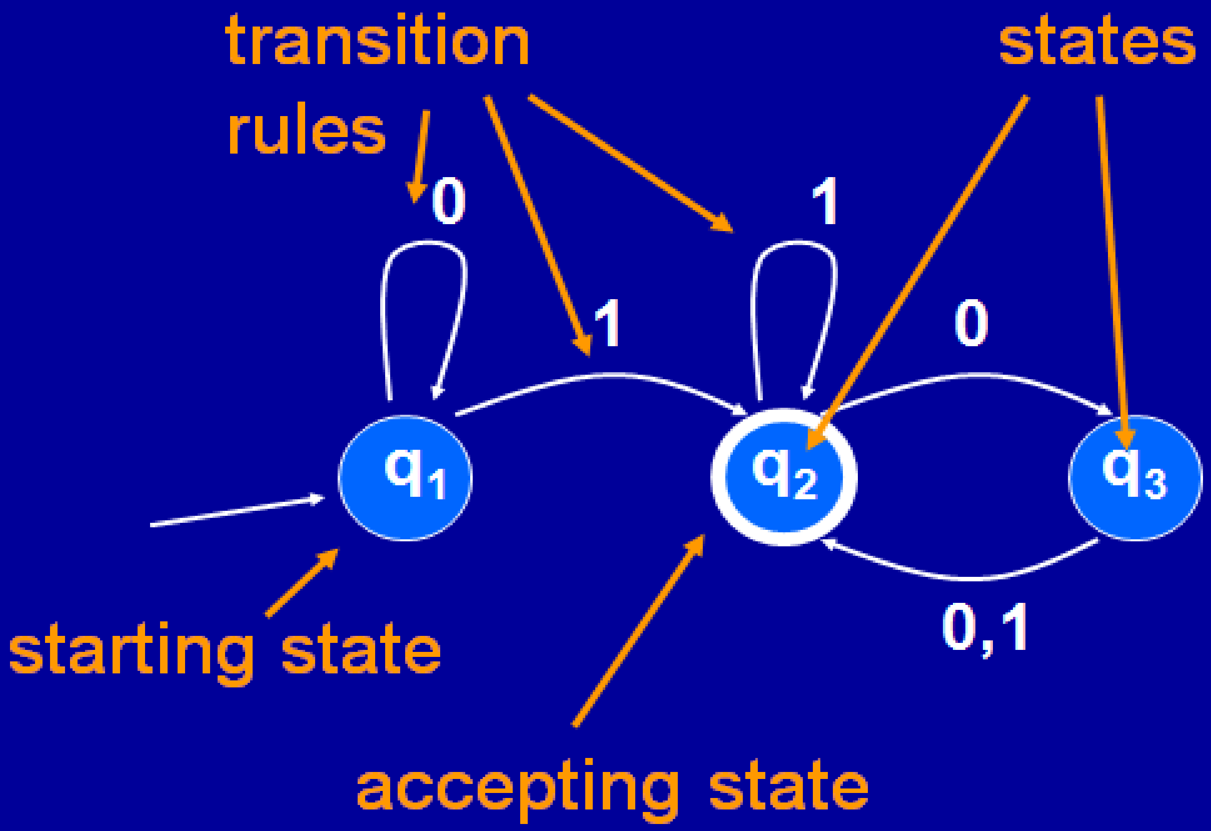
Finite Automata FA

- It started as a simple automatic device with no memory.

We still need it to study how to model a device and model its capability.

- Its goal is to act as a recognizer for specific a language/pattern.
- Any problem can be presented in form of a decidable problem that can be answered by Yes/No.
- A problem can be concatenated to one of its possible solutions and can be seen as a string that matches a pattern.
- Hence FA (machine with limited memory) can solve any problem.

A Simple Automaton (0)



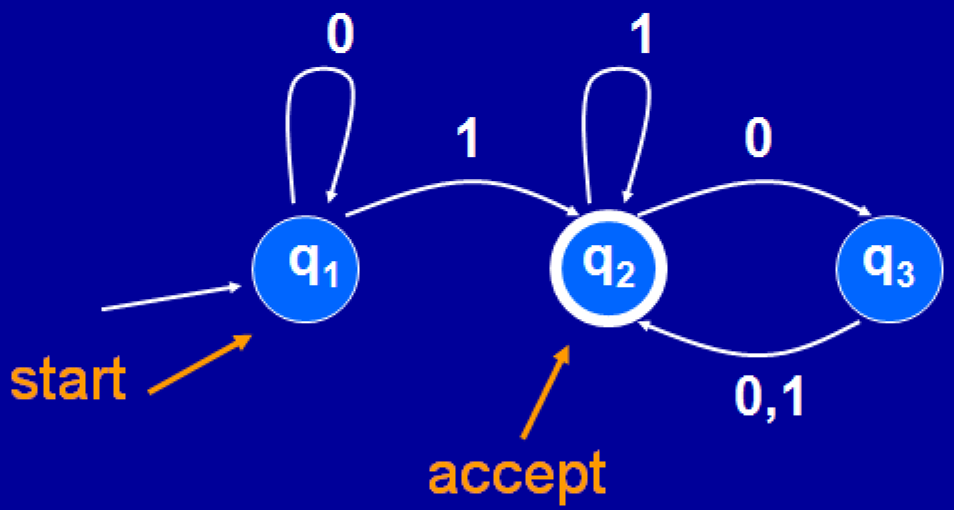
transition rules

states

starting state

accepting state

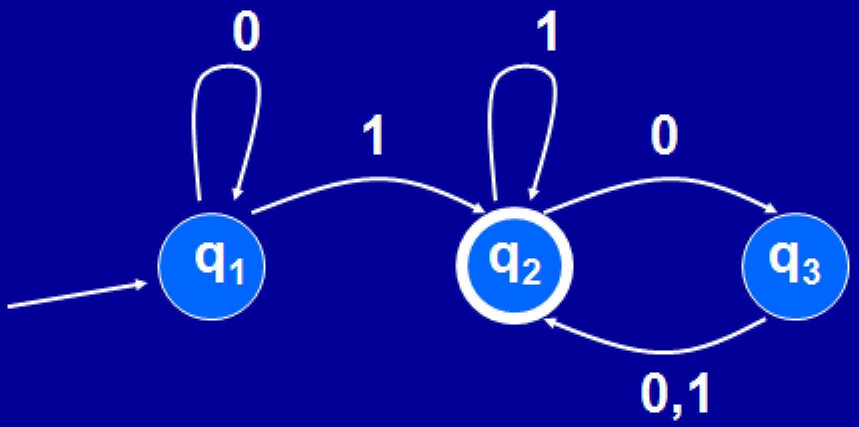
A Simple Automaton (1)



on input “0110”, the machine goes:

$q_1 \rightarrow q_1 \rightarrow q_2 \rightarrow q_2 \rightarrow q_3$ = “reject”

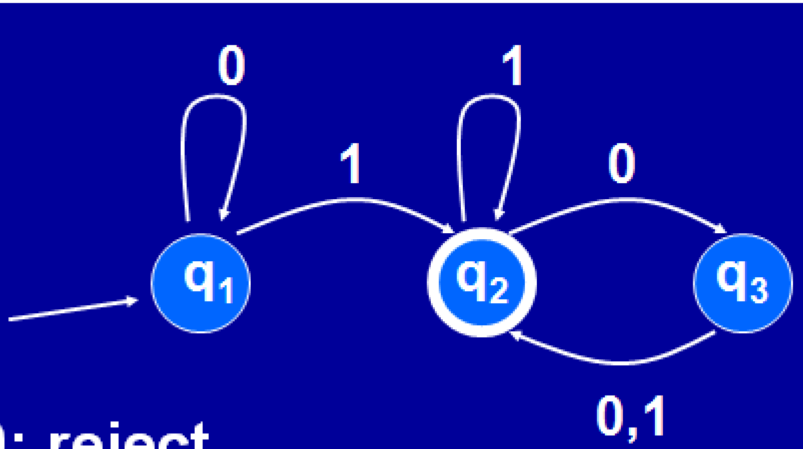
A Simple Automaton (2)



on input “101”, the machine goes:

$q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_2$ = “accept”

A Simple Automaton (3)



010: reject

11: accept

010100100100100: accept

010000010010: reject

ε: reject

Finite Automaton (def.)

- A deterministic finite automaton (DFA) M is defined by a 5-tuple $M=(Q,\Sigma,\delta,q_0,F)$
  - Q: finite set of states
  - Σ: finite alphabet
  - δ: transition function $\delta:Q\times\Sigma\rightarrow Q$
  - $q_0\in Q$: start state
  - $F\subseteq Q$: set of accepting states

Deterministic Finite Automata DFA

□ FA = "a 5-tuple " ➔ δ(Q, Σ,, q_0, F)

1.Q: {q_0, q_1, q_2, …} is set of states.

2.Σ: {a, b, …} set of alphabet.

3.(delta)δ: represents the set of transitions that FA can take between its states.

: Q x δΣ→Q

Q x Σ to Q, this function:

□Takes a state and input symbol as arguments.

□**Returns a single state**.

4. $q_0 \in Q$ is the start state.

5. $F \subset Q$ is the set of final/accepting states.

How does FA work?

1. Starts from a start state.
2. Loop

   Reads a letters from the input
3. Until input string finishes
4. If the current state is a final state then

   Input string is accepted.
5. Else

   Input string is NOT accepted.

□ **But how can FA be designed and represented?**

The Transition Function

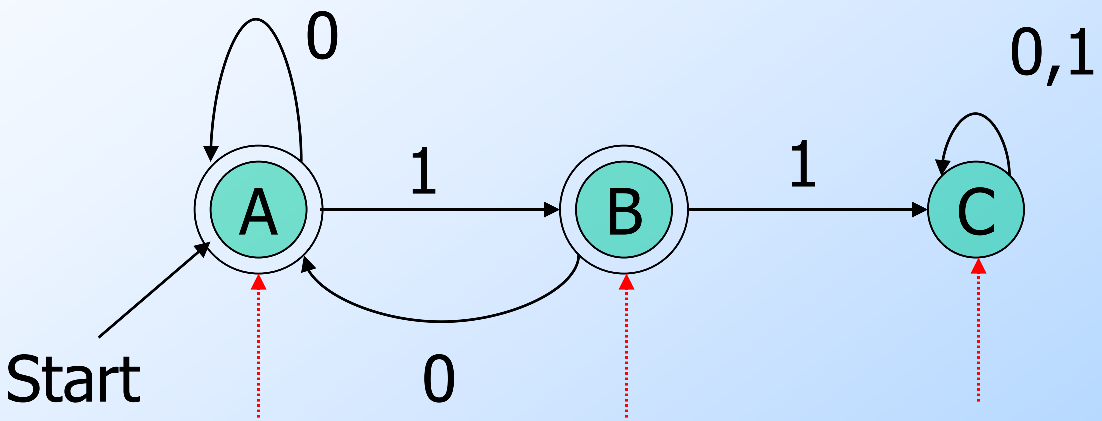
- Takes two arguments: a state and an input symbol.
- δ(q, a) = the state that the DFA goes to when it is in state *q* and input *a* is received.

Graph Representation of DFA’s

- Nodes = states.
- Arcs represent transition function.
  - Arc from state p to state q labeled by all those input symbols that have transitions from p to q.
- Arrow labeled “Start” to the start state.
- Final states indicated by double circles.

Example: Graph of a DFA

Accepts all strings without two consecutive 1's.



Previous string OK, does not end in 1.

Previous String OK, ends in a single 1.

Consecutive 1's have been seen.

Alternative Representation: Transition Table

Final states starred

Columns = input symbols

	0	1
→ * A	A	B
* B	A	C
C	C	C

Arrow for start state

Rows = states

L={w/w is of even length & begins with 01}

Exercise 1

DFA that accepts all and only the strings of 0’s and 1’s that have the sequence “ 01” some where in the string

Extended Transition Function

- We describe the effect of a string of inputs on a DFA by extending δ to a state and a string.
- Induction on length of string.
- Basis: $\delta(q, \epsilon) = q$
- Induction: $\delta(q,wa) = \delta(\delta(q,w),a)$
  - w is a string; a is an input symbol.

Extended δ: Intuition

- Convention:
  - … w, x, y, x are strings.
  - a, b, c,… are single symbols.
- Extended δ is computed for state q and inputs $a_1a_2…a_n$ by following a path in the transition graph, starting at q and selecting the arcs with labels a_1, a_2,…,a_n in turn.

Example: Extended Delta

	0	1
A	A	B
B	A	C
C	C	C

$$\delta(B,011) = \delta(\delta(B,01),1) = \delta(\delta(\delta(B,0),1),1) =$$

$$\delta(\delta(A,1),1) = \delta(B,1) = C$$

Delta-hat

- In book, the extended δ has a "hat" to distinguish it from δ itself.
- Not needed, because both agree when the string is a single symbol.
- $\hat{\delta}(q, a) = \delta(\hat{\delta}(q, \epsilon), a) = \delta(q, a)$

Extended deltas

Language of a DFA

- Automata of all kinds define languages.
- If A is an automaton, L(A) is its language.
- For a DFA A, L(A) is the set of strings labeling paths from the start state to a final state.
- Formally: L(A) = the set of strings w such that $\delta(q_0, w)$ is in F.

Exercise 2

L={w/w is ends in a 1}

Exercise 3

Construct DFA for the following languages when ∑={0,1}

a)Beginning with 101

Exercise 3

Construct DFA for the following languages when ∑={0,1}

b) Ending with 101

c)Containing 101 as substring

d)Which doesn’t contain 101 as substring

Exercise 3

Construct DFA for the following languages when ∑={0,1}

c)Containing 101 as substring

d)Which doesn’t contain 101 as substring

Exercise 3

Construct DFA for the following languages when ∑={0,1}

d)Which doesn’t contain 101 as substring

Exercise-4

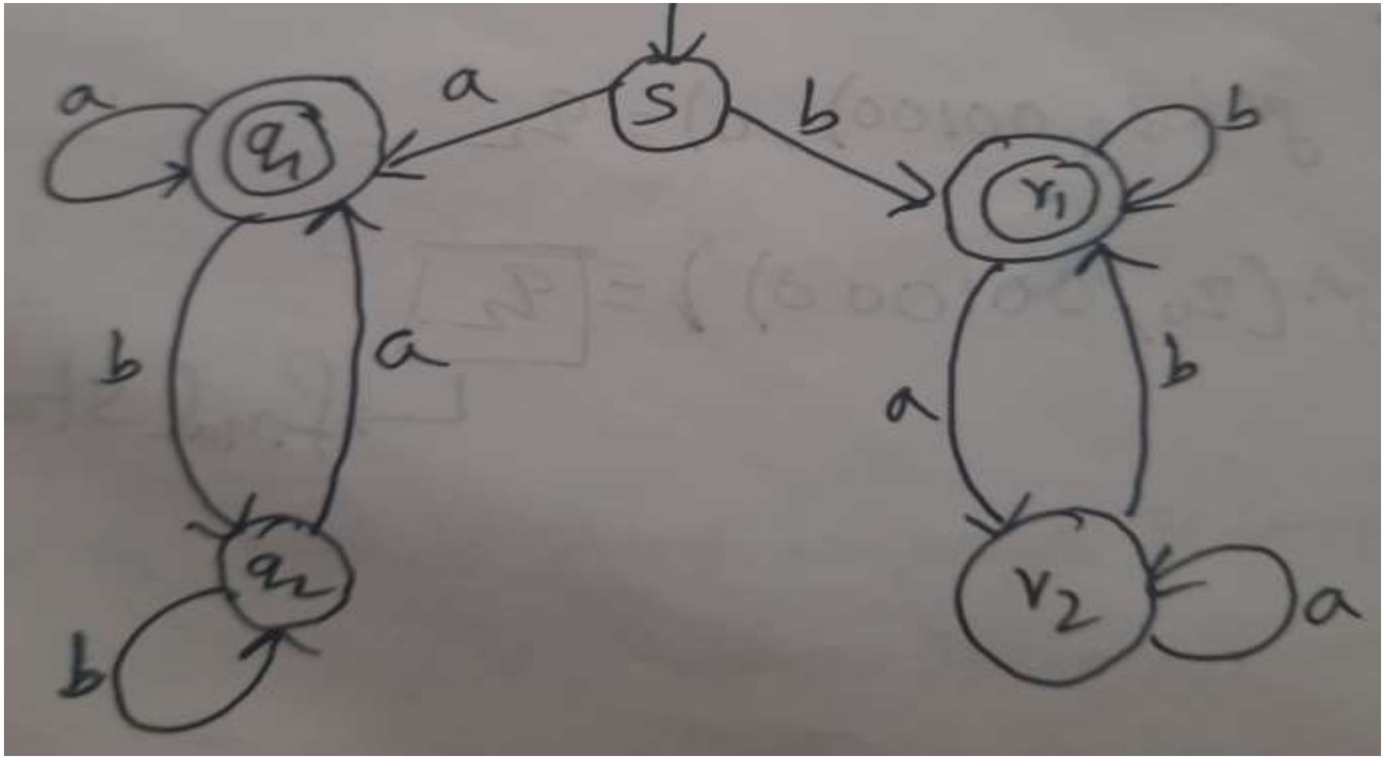
L={W/W if of even length & begins with 01}

Exercise-5

Construct DFA, that accepts set of all strings over∑= {a,b}

a) Length 2 only
b) Length with atleast 2
c) Length with atmost 2
d) Strings of even length
e) Strings of Odd length

Write the language described by the following automata



Exercise 6:Construct DFA of a number whose binary representation is divisible by 3

Exercise 6:Construct DFA of a number whose binary representation is divisible by 5

Exercise 7

- Construct the DFA which starts with and ends with different symbol
- Construct the DFA which starts with and ends with same symbol

Exercise 8

Construct DFA in which every ‘a’ is followed by bb

Exercise 7

Construct the DFA which starts with and ends with different symbol

Exercise 8

Construct DFA in which every ‘a’ is followed by bb

Exercise 8.1

Construct DFA in which atleast one ‘a’ is followed by bb

Exercise 9

Construct DFA in which every ‘a’ is never followed by bb

- Construct DFA which accepts strings a^n b^m where n,m >= 1
- Construct DFA which accepts strings a^n b^m where n,m >= 0

Exercise 10

Construct DFA which accepts strings a^n b^m where n,m >= 1

Exercise 11

Construct DFA which accepts strings a^n b^m where n,m >= 0

Exercise 12

Construct DFA which accepts strings a^n b^m c^l where n,m,l>= 1

Exercise 13

Construct DFA which accepts strings over ∑={a,b} in which the following should be accepted

a)Even no.of a’s Even no.of b’s

b)Even no.of a’s and Odd no.of b’s

c)Odd no.of a’s and Even no.of b’s

d) Odd no.of a’s and Odd no.of b’s

Exercise 13

Construct DFA which accepts strings over ∑={a,b} in which the following should be accepted

a)Even no.of a’s Even no.of b’s

b)Even no.of a’s and Odd no.of b’s

c)Odd no.of a’s and Even no.of b’s

d) Odd no.of a’s and Odd no.of b’s

Exercise 14

Construct the DFA over ∑={a,b} in which no.of a’s divisible by 3 and no.of b’s divisible by 2

Exercise 14

Construct the DFA over ∑={a,b} in which no.of a’s divisible by 3 and no.of b’s divisible by 2

Exercise 15

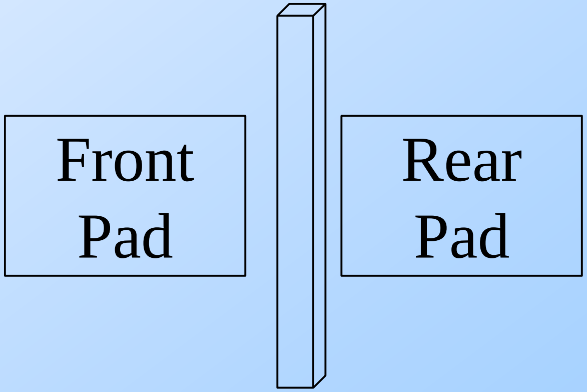
Construct the DFA over ∑={a,b} in which no.of a’s divisible by 3 and no.of b’s divisible by 3

Exercise 16

Construct the DFA over ∑={0,1} when interpreted in reverse order as binary integer is divisible by 5

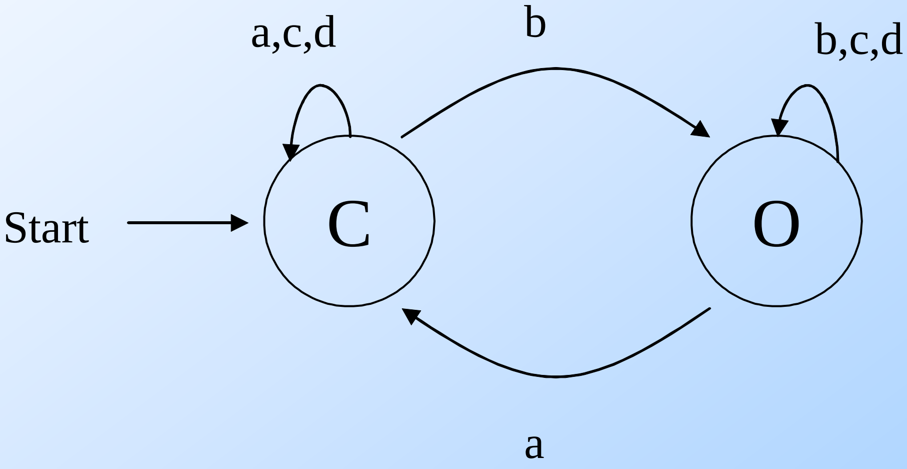
Simple Example – 1 way door

- As an example, consider a one-way automatic door. This door has two pads that can sense when someone is standing on them, a front and rear pad. We want people to walk through the front and toward the rear, but not allow someone to walk the other direction:



One Way Door

- Let's assign the following codes to our different input cases:
  - a - Nobody on either pad
  - b - Person on front pad
  - c - Person on rear pad
  - d - Person on front and rear pad
- We can design the following automaton so that the door doesn't open if someone is still on the rear pad and hit them:


a,c,d
b
b,c,d
Start
C
O
a