

If you can keep your head
when all about you are losing theirs,

If you can keep your head
when all about you are losing theirs,
its just possible you haven't grasped
the situation.

If you can keep your head
when all about you are losing theirs,
its just possible you haven't grasped
the situation.

Rose Fitzgerald Kennedy

What *is* the situation

- 5 days ago you were given an assignment to write an 11-line program (+ tests)
- It's in a language that you've never used before
- based on a concept (algebras) that you have forgotten about
- You saw some example programs in class, but you didn't take notes.

CS311—Computational Structures

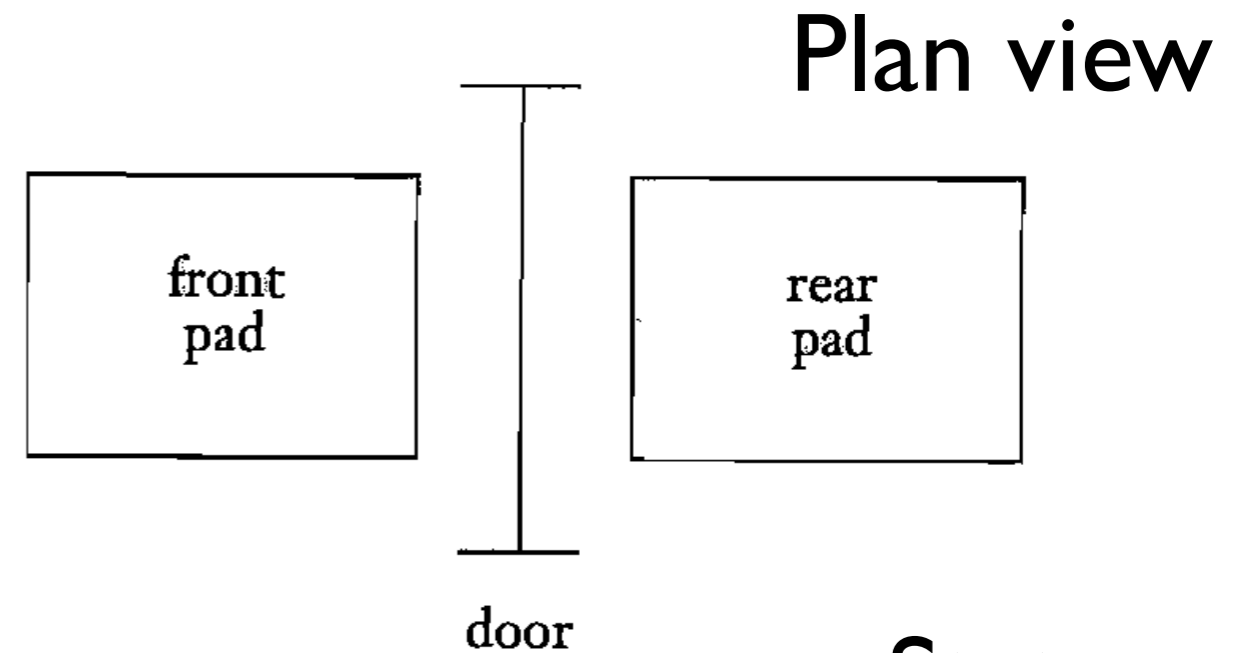
Finite State Automata

Deterministic Finite State Automata

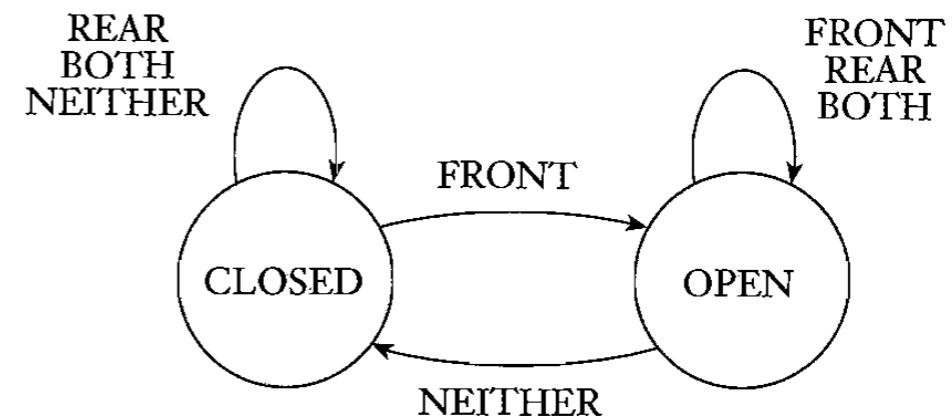
- A very simple form of “computer”
- Used in real life for control circuits
 - Hardware control: door circuits
 - Software control: telephone and network communications

Example: Door Controller

- As found at supermarket or airport
- The state diagram is a universally-understood way of describing such a machine.



State Diagram

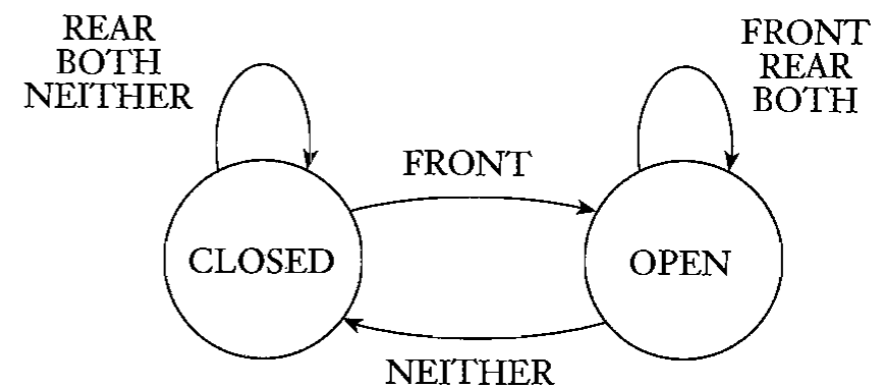


Door Controller (continued)

- This FSA can also be represented as a transition function or transition table:

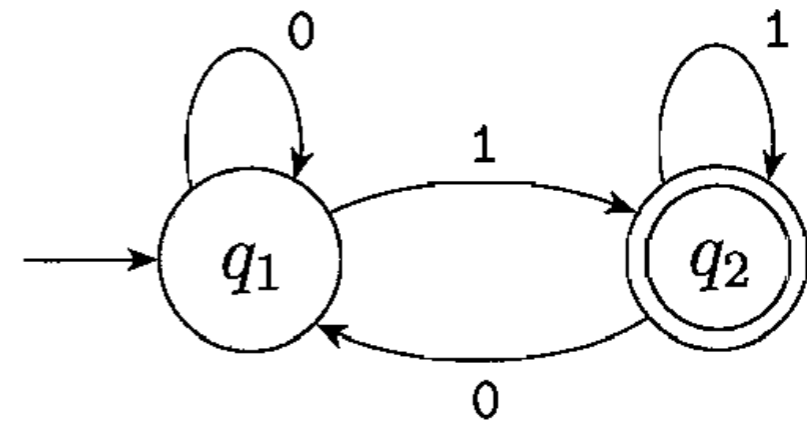
		input signal			
		NEITHER	FRONT	REAR	BOTH
state	CLOSED	CLOSED	OPEN	CLOSED	CLOSED
	OPEN	CLOSED	OPEN	OPEN	OPEN


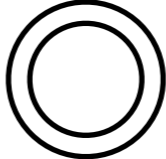
- This contains the *same* information as the diagram



FSA that “recognize” languages

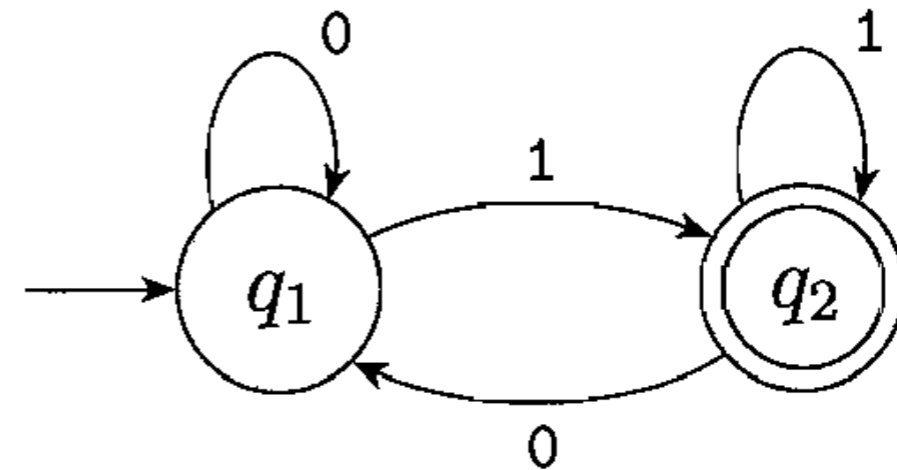
- FSA “accepts” a string if it ends up in a “final” state after reading that string from an “input tape”.



- Start state indicated with 
- Final states indicated with 
- What language is accepted by the FSA in the figure?

Let's try some examples

- 0
- 1
- 01
- 10
- 001
- 110



Formal Definition of DFA

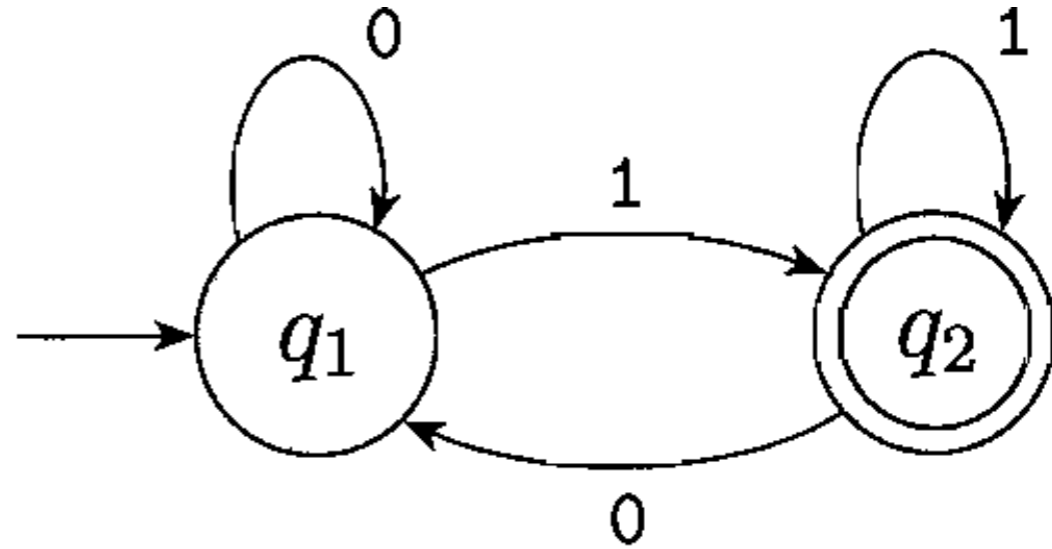
- A (deterministic) finite automaton is a 5-tuple (Q, A, δ, q_0, F) where:
 1. Q is a finite set called the **states**,
 2. A is a finite set called the **alphabet**,
 3. $\delta: Q \times A \rightarrow Q$ is the **transition function**,
 4. q_0 is the **start state**, and
 5. $F \subseteq Q$ is the set of **final states**

Why use a formal definition?

1. It is precise, e.g., it says that
 1. There can be no accept states ($F = \emptyset$)
 2. δ is total, so there is *exactly one* “next state” for each input symbols in the Alphabet
2. We can easily turn it into a computer program

Example

- Diagram:
- Formal definition:



1. $Q = \{ \quad \}$

2. $A = \{ \quad \}$

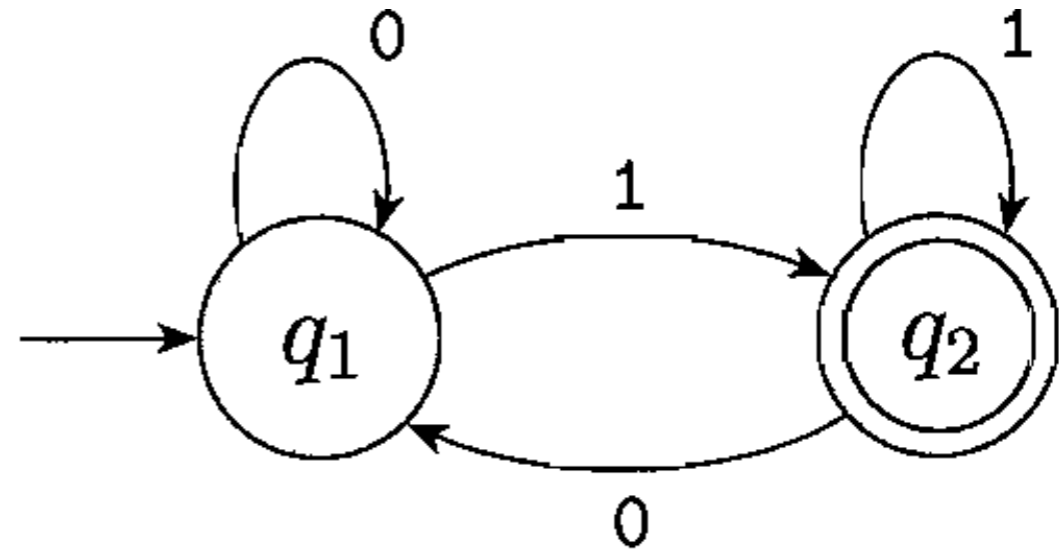
3. $q_0 =$

4. $\delta =$

5. $F = \{ \quad \}$

Example, continued

- What strings are accepted by this FSM?
- The set of all strings accepted by a FSM forms the language *recognized* by the FSM.



$$L = \{ w \in \{0,1\}^* \mid \quad \quad \quad \}$$

FSMs for Regular Languages

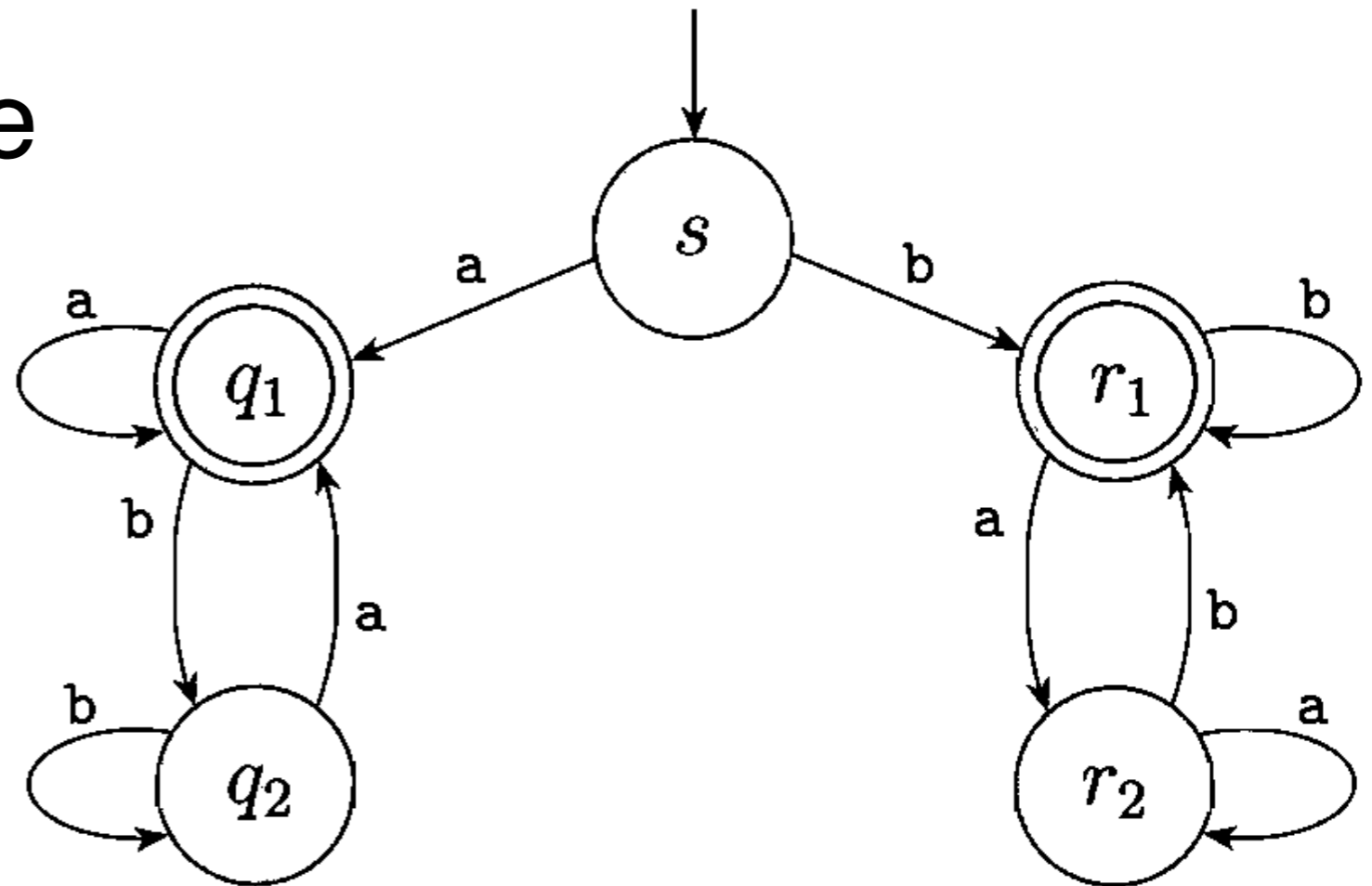
- What FSM recognizes the language \emptyset ?
- What FSM recognizes the language Λ ?

- What FSM recognizes the language $\{a\}$?

- What FSM recognizes the language $\{a\}$?
- What FSM recognizes the language $\{a\}$ over the alphabet $\{a, b\}$?

Another Example

- What language is recognized by this machine?



Formal Definition of FSM Computation

- Let $M = (Q, A, \delta, q_0, F)$ be a FSM and let $w = w_1w_2\dots w_n$ be a string, where each $w_i \in A$.
- M accepts w if there is a sequence of states $r_0, r_1, r_2, \dots, r_n \in Q$ such that:
 1. $r_0 = q_0$
 2. $\delta(r_{i-1}, w_i) = r_i$ for $i = 1, 2, \dots, n$
 3. $r_n \in F$

Language Recognition

- A machine M recognizes a language L if $L = \{ w \mid M \text{ accepts } w \}$

ML definition of a FSM

```
datatype 'a Label = Label of 'a;
```

```
datatype State = State of string;
```

```
datatype 'a FSM = FSM of  
  { allStates : State list,  
    alphabet : 'a list,  
    transitions : (State * 'a Label * State) list,  
    startState : State,  
    finalStates : State list };
```

(* This defines the type FSM as a structure with five named fields. *)

(* For convenience, let's define five projection functions that extract the fields. *)

```
fun allStates (FSM {allStates, alphabet, transitions, startState, finalStates}) = allStates;  
fun startState (FSM {allStates, alphabet, transitions, startState, finalStates}) = startState;  
fun alphabet (FSM {allStates, alphabet, transitions, startState, finalStates}) = alphabet;  
fun transitions (FSM {allStates, alphabet, transitions, startState, finalStates}) = transitions;  
fun finalStates (FSM {allStates, alphabet, transitions, startState, finalStates}) = finalStates;
```

(* We are going to use lists to represent sets, so we need to implement the set operation union, the set membership test, and an operation that adds a single element to the set. We also define a function setOf that turns an arbitrary list into a "set list" (by removing duplication) *)

```
fun union (l1: 'a list, []: 'a list) = l1  
  | union ([], l2) = l2  
  | union (e::es, l2) = union (es, add e l2)  
  
and member e l = List.exists (fn x => x = e) l  
  
and add e l = if member e l then l else e::l;  
  
fun setOf [] = []  
  | setOf [e] = [e]  
  | setOf (e::es) = add e (setOf es);
```

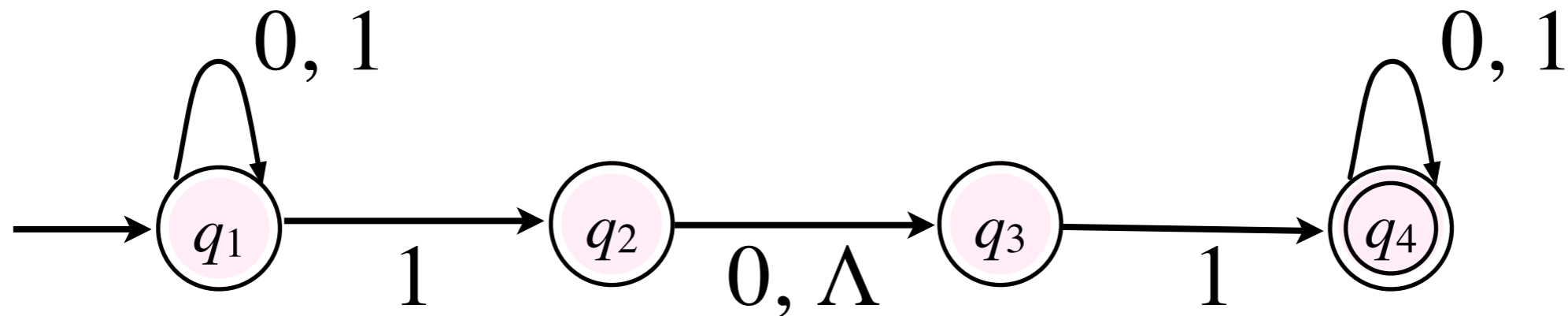
Amazing Theorem

- The class of regular languages and the class of languages accepted by some FSM are identical.

Nondeterminism

- In the FSA that we have seen so far, there is exactly one action to be taken on each input symbol.
 - that's what it means for δ to be a function!
- In a nondeterministic FSA, several choices may exist at each step.

Example of NFA



- How does it differ from a DFA?
 1. some states have multiple transitions on a given input
 2. some states have *no* transition on an input
 3. some transitions have label Λ

Formal Definition

- A nondeterministic finite automaton is a 5-tuple (Q, A, δ, q_0, F) where:
 1. Q is a finite set called the **states**,
 2. A is a finite set called the **alphabet**,
 3. $\delta: Q \times (A \cup \{\Lambda\}) \rightarrow \mathcal{P}(Q)$ is the **transition function**,
 4. q_0 is the **start state**, and
 5. $F \subseteq Q$ is the set of **final states**

Nondeterministic Computation

- What does it mean for an NFA to take a “step” when there are multiple possibilities at each step? What about Λ ?
 - the NFA makes all possible transitions in parallel, or, equivalently,
 - the NFA clones itself and one clone explores each possibility.
- an NFA can reach a “dead end”
- an NFA accepts its input if any of the clones reaches a final state.