CS311—Computational Structures

Finite State Automata

Lecture 2

Andrew P. Black
Andrew Tolmach

Portland State University

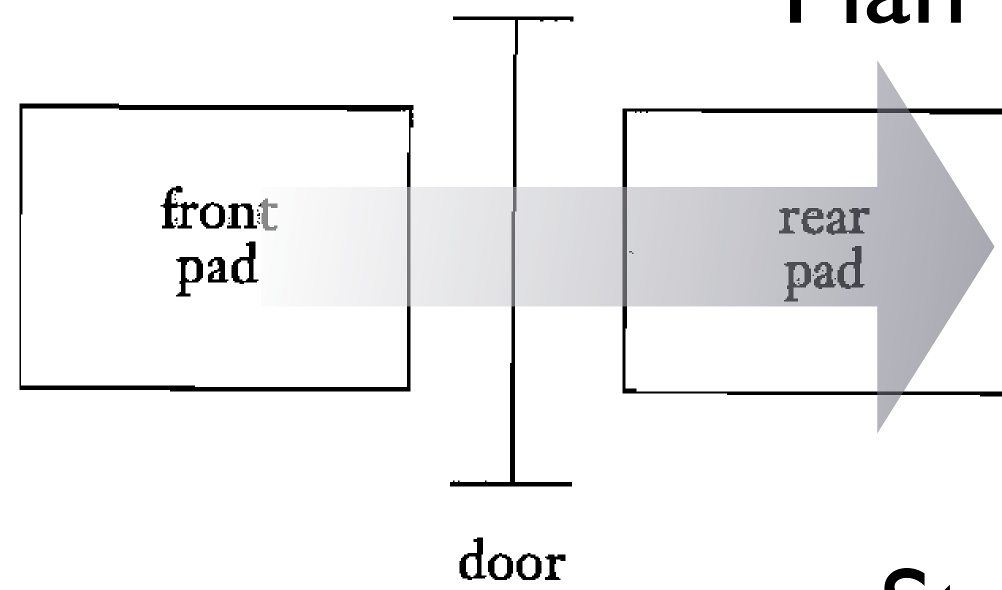
Deterministic Finite State Automata

- A very simple form of "computer"
- Used in real life for control circuits
  - Hardware control: e.g. traffic lights, appliances, computer CPU's
  - Software control: e.g. servers, games, telephone and network communications

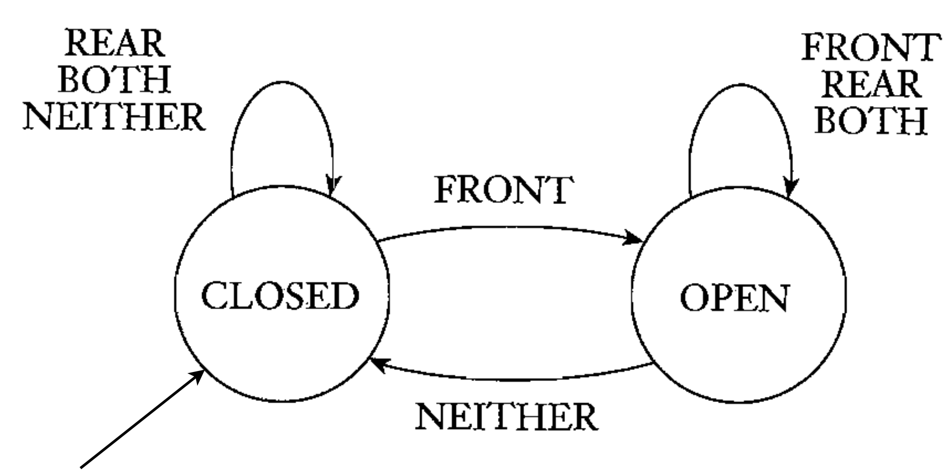
Example: Door Controller

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

Plan view



State Diagram



Portland State UNIVERSITY

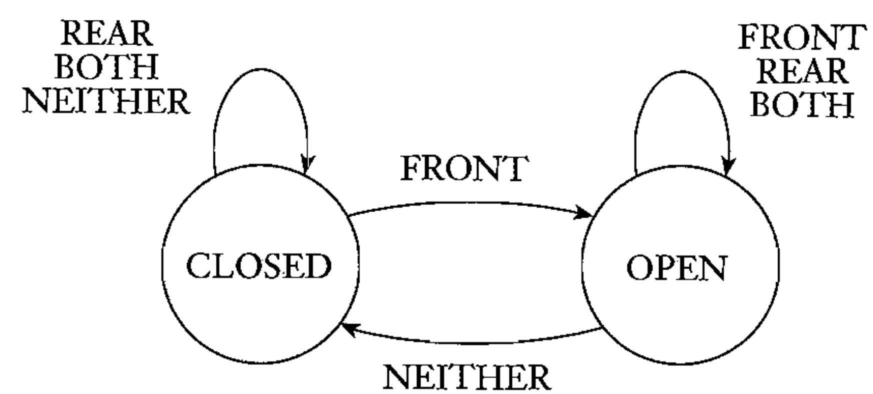
Door Controller (continued)

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

input signal

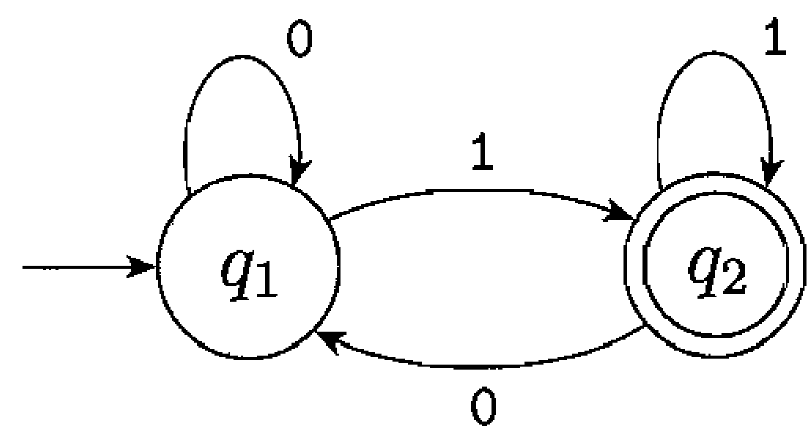
state		NEITHER	FRONT	REAR	BOTH
	CLOSED	CLOSED	OPEN	CLOSED	CLOSED
	OPEN	CLOSED	OPEN	OPEN	OPEN

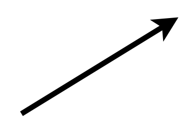
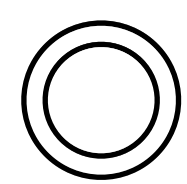
- This contains the *same* information as the diagram



FSA that "recognize" languages

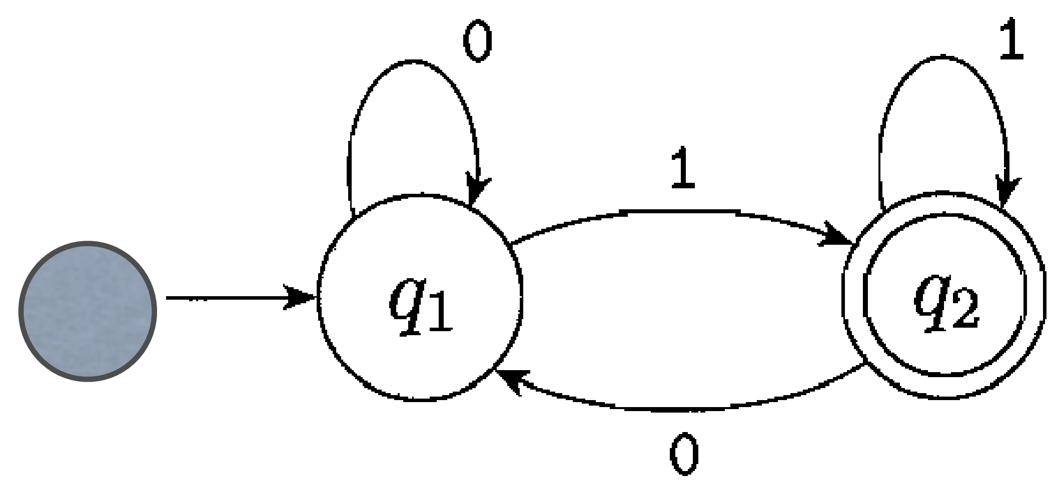
- FSA "accepts" a string if it ends up in a "final" state after reading that string from an "input tape".



- Start state indicated with 
- Final states indicated with 
- What strings are accepted by the FSA in the figure?

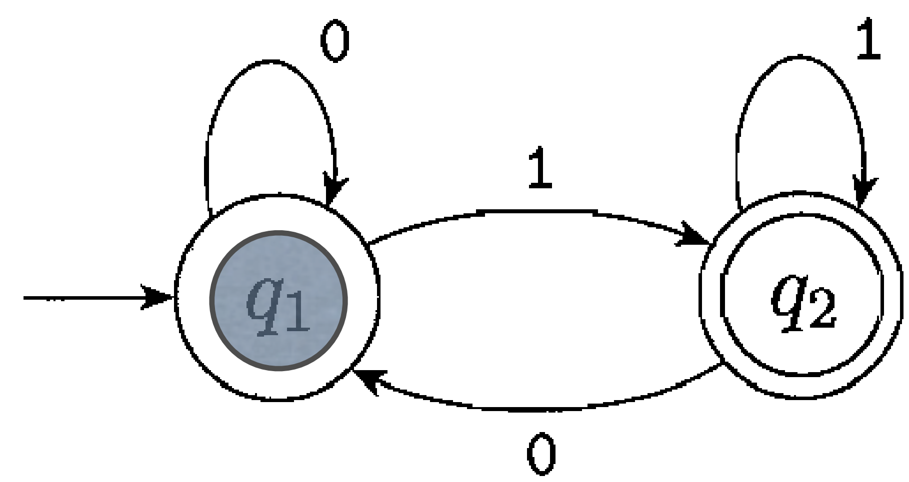
Let's try an example

- input: 100101



Let’s try an example

- input: 100101

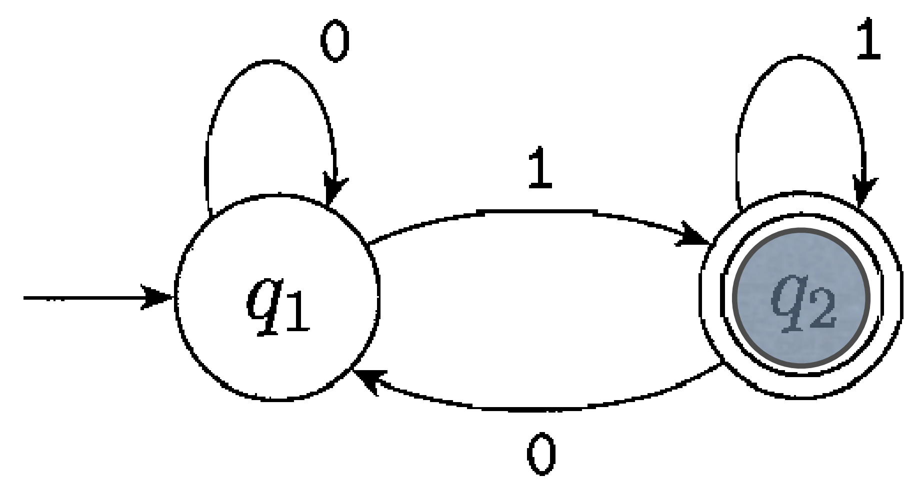


Always start in state q_1

Portland State University

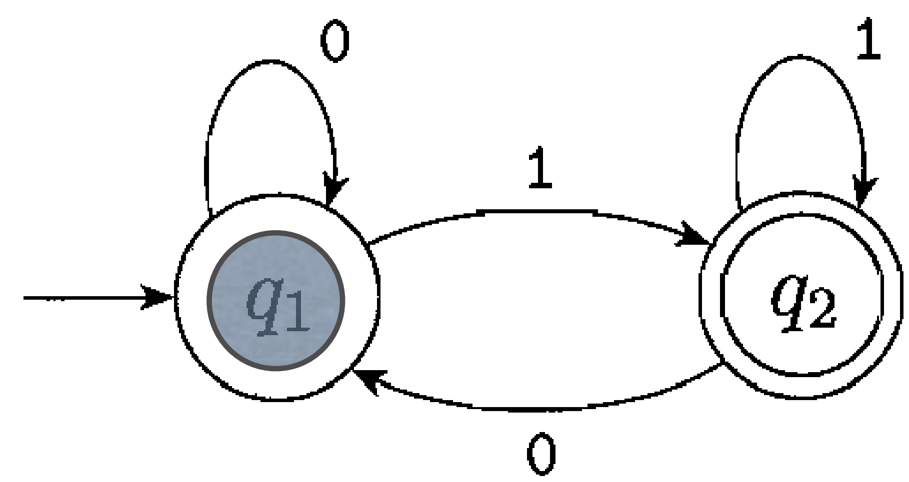
Let's try an example

- input: 100101



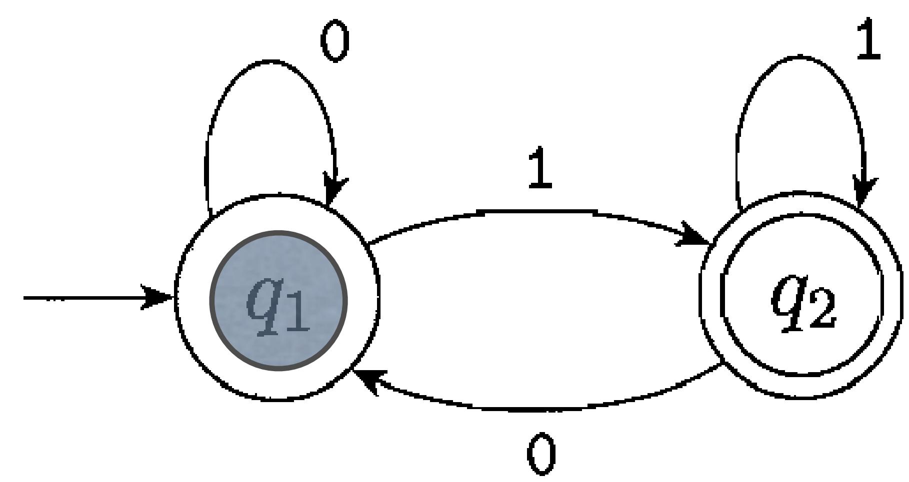
Let’s try an example

- input: 100101



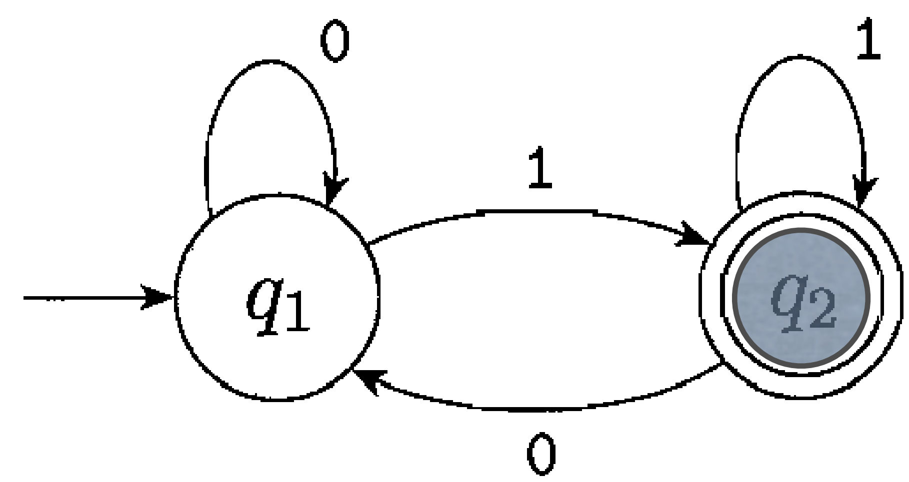
Let's try an example

- input: 100101



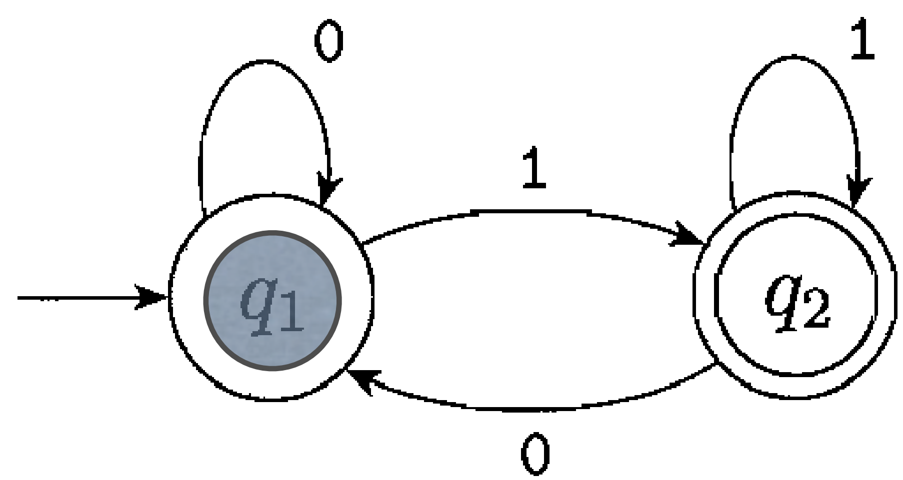
Let's try an example

- input: 100101



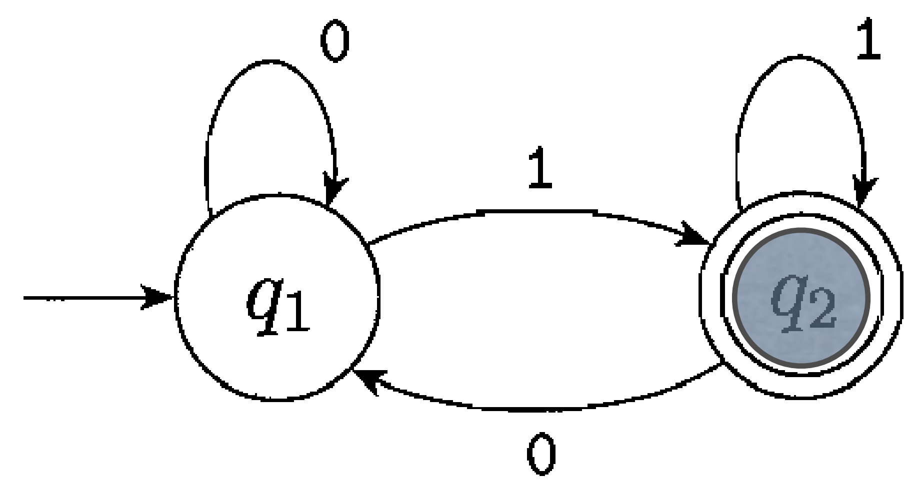
Let's try an example

- input: 100101



Let's try an example

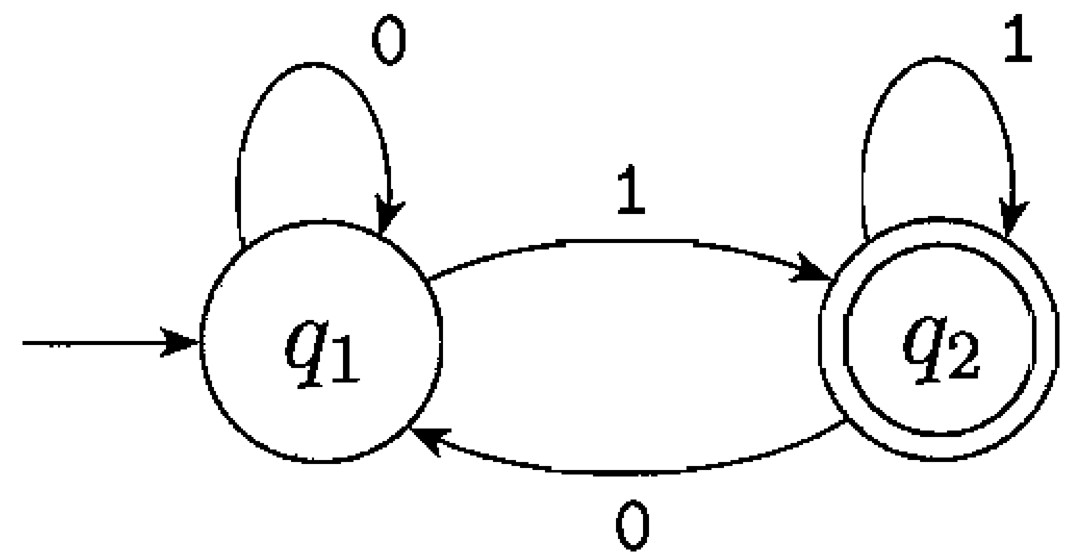
- input: 100101



Since machine is in a final state when it reaches the end of the input, it ACCEPTS the input

Example, continued

- What strings are accepted by this DFA?
- The set of all strings accepted by a DFA forms the language accepted (or *recognized)* by the DFA.



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

Formal Definition of DFA

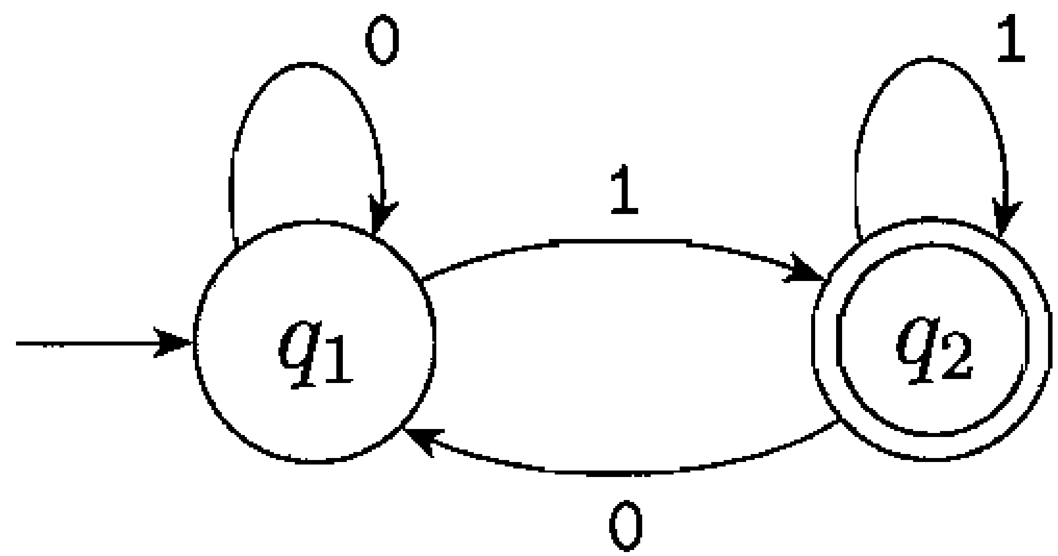
- A (deterministic) finite (state) automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

1. Q is a finite set called the **states**,
2. Σ is a finite set called the **alphabet**,
3. $\delta: Q \times \Sigma \to Q$ is the **transition function**,
4. $q_0 \in Q$ is the **start state**, and
5. $F \subseteq Q$ is the set of **final** (or **accept**) 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 prove things about it.
3. We can easily turn it into a computer program

Example, again



- Diagram:
- Formal definition:

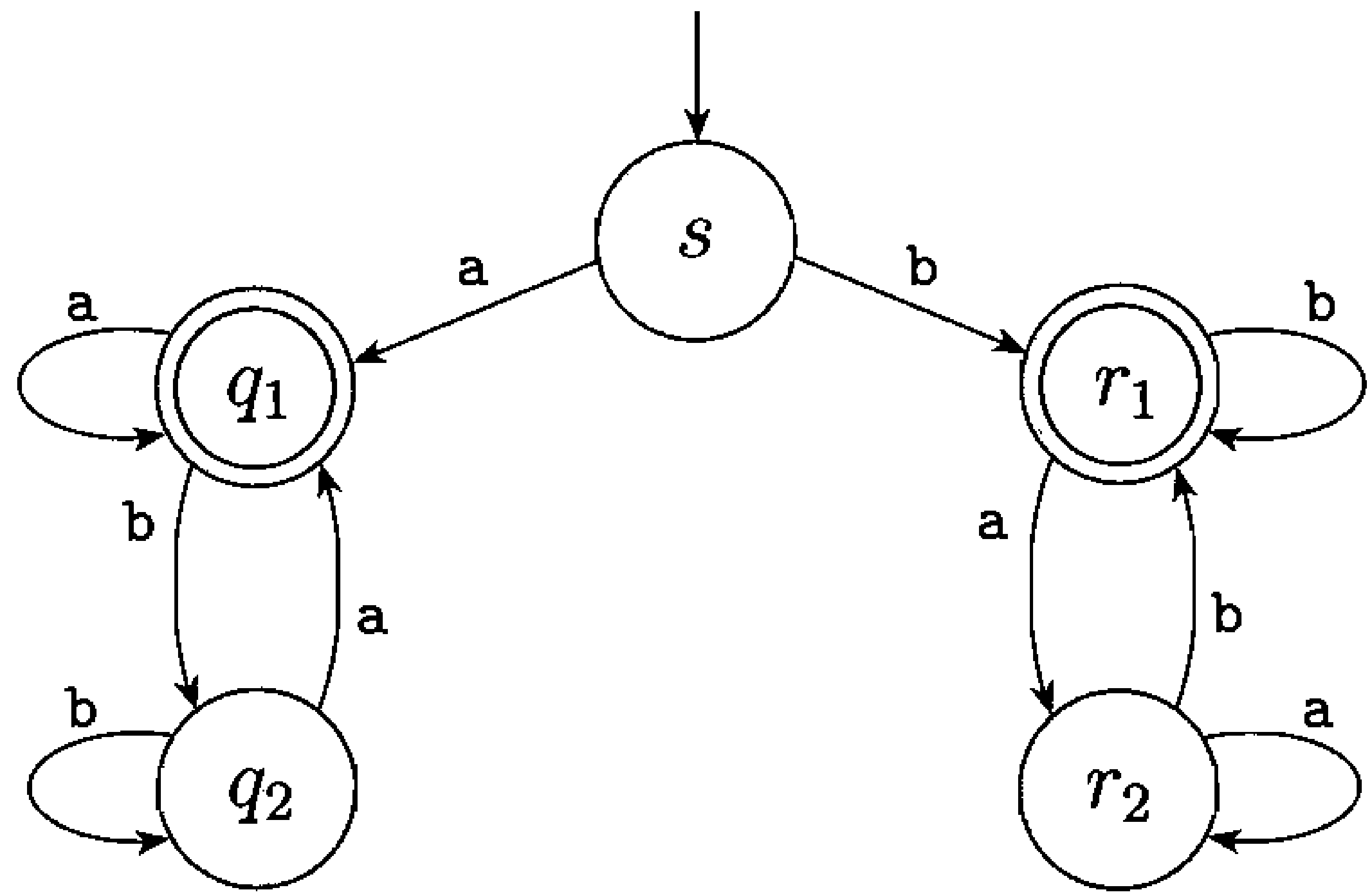
1. Q = {   }
2. Σ = {   }
3. q_0 =
4. δ =
5. F = {   }

DFAs for Simple Languages

- Consider the alphabet $\Sigma = \{a,b\}$
- What DFA recognizes the language $\varnothing$?
- What DFA recognizes the language $\{\varepsilon\}$?
- What DFA recognizes the language $\{a\}$?
- The language $\{aa\}$? The language $\{a,b\}$? The language $\{aa,ab\}$?

Another Example

- What language is recognized by this machine?



- Stumped? Try using a simulator tool to explore the machine's behavior on different inputs. (See course web page for a few pointers.)

Formal Definition of DFA Computation

- Let M = (Q, Σ, δ, q_0, F) be a DFA and let $w = a_1 a_2 \ldots a_n$ be a string, where each $a_i \in \Sigma$.
- M accepts w iff there is a sequence of states $r_0, r_1, r_2, \ldots, r_n \in Q$ such that:

1. $r_0 = q_0$
2. $r_i = \delta(r_{i-1}, a_i)$ for $i = 1, 2, \ldots n$
3. $r_n \in F$

Portland State University

IALC's Definition of Acceptance

Extend the definition of δ (which is defined on symbols) to $\hat{\delta}$, defined on strings of symbols:

$$\begin{aligned} \hat{\delta}(q, \epsilon) &= q \\ \hat{\delta}(q, xa) &= (\delta(\hat{\delta}(q, x), a) \qquad \forall a \in \Sigma, x \in \Sigma^* \end{aligned}$$

Now we say that M accepts string w iff $\hat{\delta}(q_0, w) \in F$.

It should be easy to see that these two definitions are equivalent, with $r_i = \hat{\delta}(q_0, a_1 a_2 a_3 \ldots a_i), \forall i \in [0, n]$

Portland State UNIVERSITY

Regular Languages

- A language L is **regular** iff there exists a DFA M such that M recognizes L.
- We write L(M) for the language recognized by M.
- Decision problems associated with regular languages are particularly simple

Combining DFA’s

- Fix alphabet $\Sigma = \{a,b\}$
- Find DFA’s recognizing the following:
  - $L_{bba} = \{w \mid w$ is one or more copies of bba$\}$
  - $L_{b...} = \{w \mid w$ starts with b $\}$
  - $L_{2a} = \{w \mid w$ contains an even number of a’s$\}$
  - Machines for $L_{bba} \cup L_{b...}$ and $L_{b...} \cup L_{2a}$ are easy
  - But $L_{bba} \cup L_{2a}$ is harder

Product of States

- Here's an easier example
  - L_{2a} = {w | w contains an even number of a's}
    - Machine has two states:
      - state AE: # of a's seen so far is even (accepting)
      - state AO: # of a's seen so far is odd (not accepting)
  - L_{2b} = {w | w contains an even number of b's}
    - Similarly, machine has states BE, BO
  - $L_{2ab} = L_{2a} \cup L_{2b}$
    - Machine has four states: (AE,BE), (AE,BO),

Closure Under Union

- Theorem: Suppose $L_1 = L(M_1)$ and $L_2 = L(M_2)$ for DFA's M_1 and M_2. Then there exists a machine M such that $L(M) = L_1 \cup L_2$.

- Proof Idea: M should simulate **both** M_1 and M_2, in the sense that it keeps track of which state **each** of them is in after each input character. M should accept if **either** M_1 or M_2 would accept.

Portland State UNIVERSITY

Details of Construction

- Let $M_1 = (Q_1,\Sigma,\delta_1,q_1,F_1)$, $M_2 = (Q_2,\Sigma,\delta_2,q_2,F_2)$
- Then $L(M) = L(M_1) \cup L(M_2)$ if $M = (Q,\Sigma,\delta,q_0,F)$, where
  - $Q = \{(r_1,r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$
    - Can also say Q is the **Cartesian product** $Q_1 \times Q_2$
  - $\delta((r_1,r_2),a) = (\delta_1(r_1,a), \delta_2(r_2,a))$
  - $q_0 = (q_1,q_2)$
  - $F = \{(r_1,r_2) \mid r_1 \in F_1 \vee r_2 \in F_2\}$

More on closure under union

- This construction is essentially what we did for L_{2ab}
- Eventual homework: give formal proof that this construction works
- What happens if we change "$\vee$" to "$\wedge$" in definition of F ?

Regular Operations

- Let A and B be languages. We define the following **regular operations**:
  - Union: $A \cup B = \{ x \mid x \in A \text{ or } x \in B \}$
  - Concatenation: $A \cdot B = \{ xy \mid x \in A \text{ and } y \in B \}$
  - Star: $A^* = \{x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A \}$
- Claim: the set of regular languages is **closed** under the regular operations (that's where the name comes from!)

Coding up DFA's

- DFAs are very easy to simulate on a computer
  - Direct-coded approach:
    - states are program locations
    - transitions are jumps
  - Table-driven approach:
    - fixed code works for all machines
    - change data for each machine

Direct-coded L_{2a} in C

```
#include "stdio.h"

#define ACCEPT {printf("accept\n"); return 0;}
#define REJECT {printf("reject\n"); return 0;}
#define IMPOSSIBLE {printf("invalid symbol in input\n"); return 1;}

int main (int argc, char **argv) {
  char *input = *++argv;

  goto Seven;

 Seven:
  switch (*input++) {
  case '\0': ACCEPT;
  case 'a':  goto Sodd;
  case 'b':  goto Seven;
  default:  IMPOSSIBLE;
  }

 Sodd:
  switch (*input++) {
  case '\0': REJECT;
  case 'a':  goto Seven;
  case 'b':  goto Sodd;
  default:   IMPOSSIBLE;
  }

}
```

Portland State UNIVERSITY

Direct-coded L_{bba} in C

```
...

int main (int argc, char **argv) {
  char *input = *++argv;

 Sstart:
  switch (*input++) {
  case '\0': REJECT;
  case 'a':  goto Serr;
  case 'b':  goto Sb;
  default:  IMPOSSIBLE;
  }

 Sb:
  switch (*input++) {
  case '\0': REJECT;
  case 'a':  goto Serr;
  case 'b':  goto Sbb;
  default:   IMPOSSIBLE;
  }

 Sbb:
  switch (*input++) {
  case '\0': REJECT;
  case 'a':  goto Sbba;
  case 'b':  goto Serr;
  default:   IMPOSSIBLE;
  }

 Sbba:
  switch (*input++) {
  case '\0': ACCEPT;
  case 'a':  goto Serr;
  case 'b':  goto Sb;
  default:   IMPOSSIBLE;
  }

 Serr: ...
}
```

Table-driven DFA Simulator

```
/* TABLE-DRIVEN DFA SIMULATOR */
/* Machine-specific data follows. It must be
adjusted for each different DFA to be simulated.
*/
/* Here we specify the DFA for language Lbba */

/* number of states */
#define STATES 5

/* number of symbols */
#define SYMBOLS 2

/* convert ASCII character to symbol number
0,1,2,...,SYMBOLS-1 */
#define SYMBOL_OF_CHAR(c) (c-'a')

/* these are just defined to increase legibility
in the remainder
   of the machine description */
#define Sstart 0
#define Sb 1
#define Sbb 2
#define Sbba 3
#define Serr 4
```

input symbol

Old state

	a	b
Sstart	Serr	Sb
Sb	Serr	Sbb
Sbb	Sbba	Serr
Sbba	Serr	Sb
Serr	Serr	Serr

```
int initial_state = Sstart;

int next_state[STATES][SYMBOLS] =
  { /* from Sstart */ {Serr,Sb},
    /* from Sb */     {Serr,Sbb},
    /* from Sbb */    {Sbba,Serr},
    /* from Sbba */   {Serr,Sb},
    /* from Serr */   {Serr,Serr} };

/* 0 means non-accepting; 1 means accepting */
int is_accepting_state[STATES] =
  {
    /* Sstart */ 0,
    /* Sb */     0,
    /* Sbb */    0,
    /* Sbba */   1,
    /* Serr */   0 };
```

Driver for table-driven DFA

```
/* ------------------------------------------------------------ */
/* The simulation code is identical for every DFA */

#include “stdio.h”

int main (int argc, char **argv) {
  char *input = *++argv;

  int current_state = initial_state;
  char c;
  while (c = *input++) {
    int symbol = SYMBOL_OF_CHAR(c);
    if (symbol >=0 && symbol < SYMBOLS)
      current_state = next_state[current_state][symbol];
    else {
      printf("invalid symbol in input\n");
      return 1;
    }
  }
  if (is_accepting_state[current_state])
    printf("accept\n");
  else
    printf("reject\n");
  return 0;
}
```

Portland State University