
TIME
COMPUTER SOFTWARE
The Magic Inside the Machine
Microsoft
Boss
Bill Gates

Time Magazine (1984)

Put the right kind of software into a computer, and it will do whatever you want it to. There may be limits on what you can do with the machines themselves, but there are no limits on what you can do with software.

Faster than a Busy Beaver

```
(define (c)
  (define (b n) ...)
  (+ 1 (b (* N 5))))
```

Why Study Theory?

- Basic Computer Science Issues
  - What is a computation?
  - Are computers omnipotent?
  - What are the fundamental capabilities and limitations of computers?
- Pragmatic Reasons
  - Avoid intractable or impossible problems.
  - Apply efficient algorithms when possible.
  - Learn to tell the difference.

A 2-MINUTE PROOF OF THE 2nd-MOST IMPORTANT THEOREM OF THE 2nd MILLENUIUM

by Doron Zeilberger

Written: Oct. 4, 1998

Kurt Gödel (1931)

- The (First) Incompleteness Theorem
- The most important theorem of the second millennium

- No formal logic for arithmetic can prove everything that is true
- $x^n+y^n=z^n$ has a solution only if $n=2$

Proof

1. Imagine some program halt(p,x) that answers "yes" when p(x) halts and "no" otherwise.
2. Construct the program

   alan(p) = if halt(p,p) says "yes"
   then "do nothing" forever
   otherwise answer "yes"
3. Consider the question halt(alan,alan).

Proof

1. Imagine some program halt(p,x) that answers "yes" when p(x) halts and "no" otherwise.
2. Construct the program

   alan(alan) = if halt(alan,alan) says "yes"
   then "do nothing" forever
   otherwise answer "yes"
3. Consider the question halt(alan,alan).

Proof

1. Imagine some program halt(p,x) that answers "yes" when p(x) halts and "no" otherwise.
2. Construct the program

   alan(alan) = if halt(alan,alan) says "yes"
   then "do nothing" forever
   otherwise answer "yes"
3. Consider the question halt(alan,alan).
   - If it answers "yes, it halts", in fact alan(alan) loops forever

Proof

1. Imagine some program halt(p,x) that answers "yes" when p(x) halts and "no" otherwise.
2. Construct the program

   alan(alan) = if halt(alan,alan) says "yes"
   then "do nothing" forever
   otherwise answer "yes"
3. Consider the question halt(alan,alan).
   - If it answers "no, it loops", in fact alan(alan) halts with answer "yes"

Proof

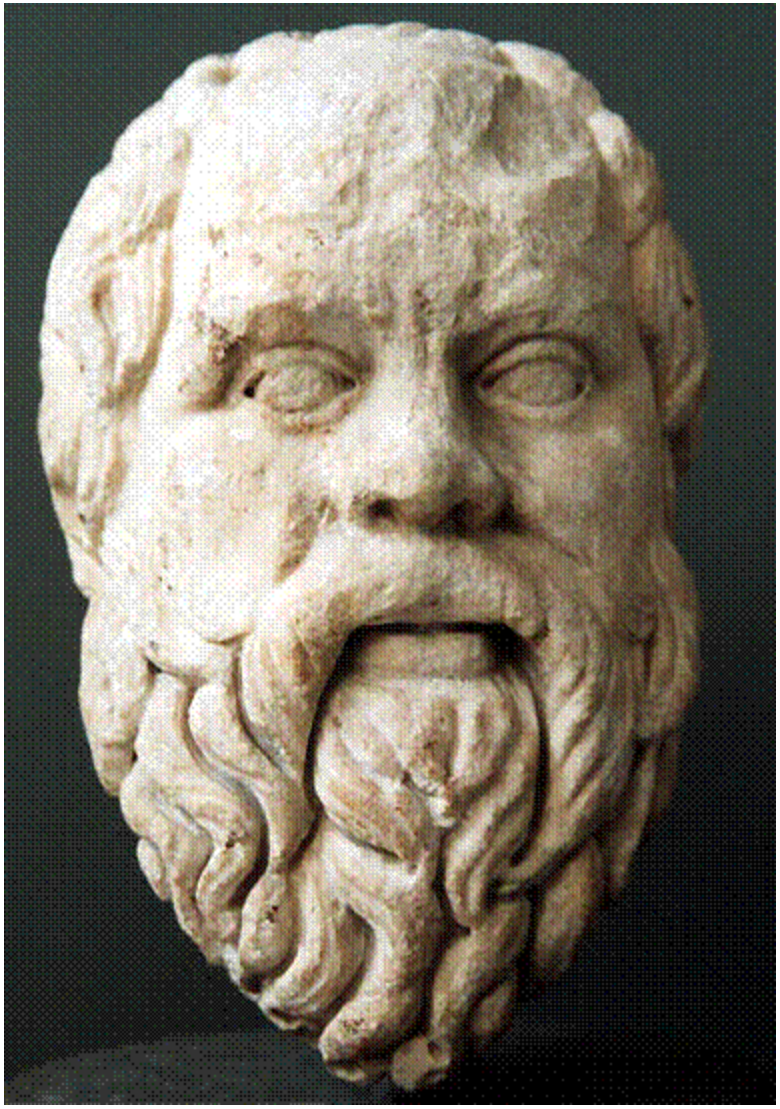
1. Imagine some program halt(p,x) that answers "yes" when p(x) halts and "no" otherwise.
2. Construct the program

   alan(alan) = if halt(alan,alan) says "yes"
   then "do nothing" forever
   otherwise answer "yes"
3. Consider the question halt(alan,alan).
   - If it never answers, then it's also no good.

Epimenides (-550)

All Cretans are liars...

One of their own poets has said so.



Socrates (-400)

One thing
that I know
is that
I know nothing

Eubulides (-350)

This statement is false!

Some Other Undecidable Problems

- Does a program run forever?
- Is a program correct?
- Are two programs equivalent?
- Is a program optimal (time wise – is it the fastest program solving a specific problem)?
- Does a polynomial equation with one or more variables and integer coefficients (*e.g.* $5x + 15y + 19xyz^2 = 12$) have an integer solution (Hilbert's 10th problem).
- Is a finitely-presented group trivial?
- Given a string, x, how compressible is it?

Critical CS Questions

What is a computer?
And What is a Computation?

Critical CS Questions

What is a computer?
And What is a Computation?

- ~~real computers too complex for any theory~~

Critical CS Questions

What is a computer?
And What is a Computation?

- ~~real computers too complex for any theory~~
- need manageable mathematical abstraction

Critical CS Questions

What is a computer?
And What is a Computation?

- ~~real computers too complex for any theory~~
- need manageable mathematical abstraction
- idealized models: accurate in some ways, but not in all details

Finite Automata

- formal definition of finite automata

Finite Automata

- formal definition of finite automata
- deterministic vs. non-deterministic finite automata

Finite Automata

- formal definition of finite automata
- deterministic vs. non-deterministic finite automata
- regular languages

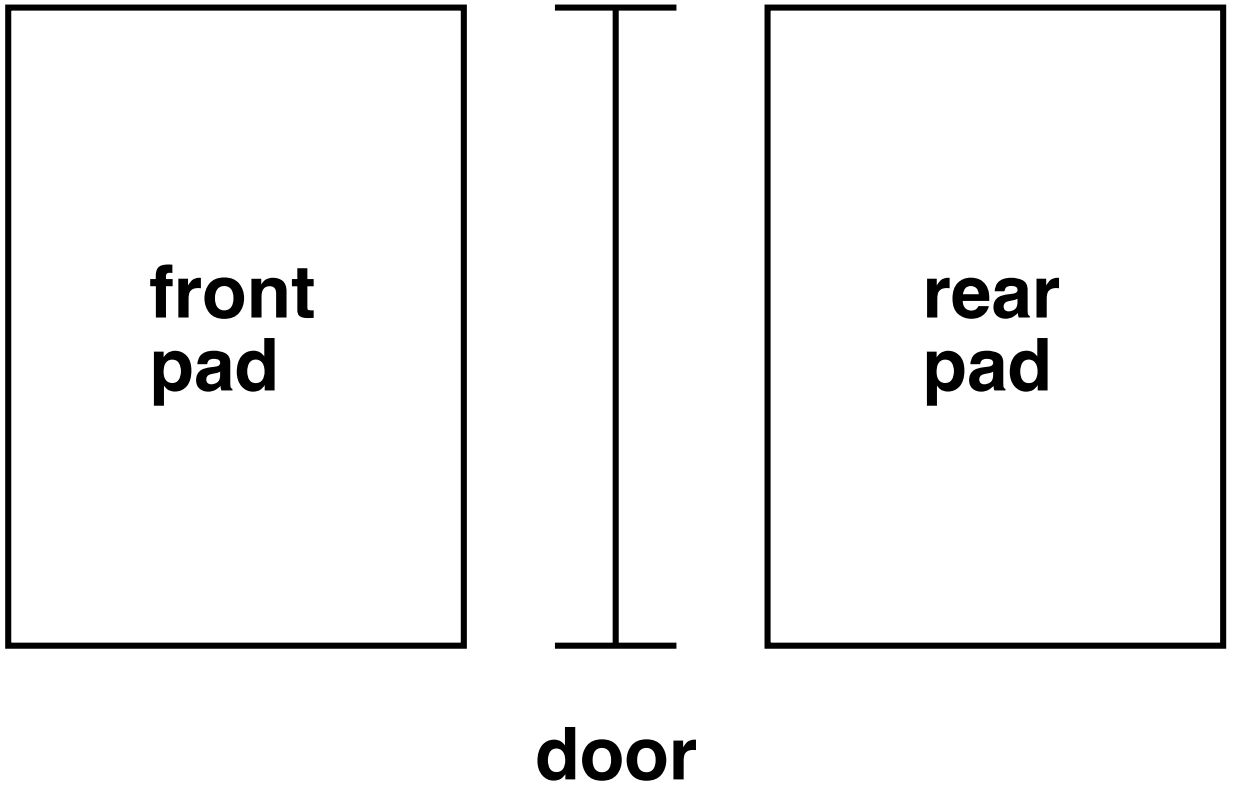
Finite Automata

- formal definition of finite automata
- deterministic vs. non-deterministic finite automata
- regular languages
- operations on regular languages

Finite Automata

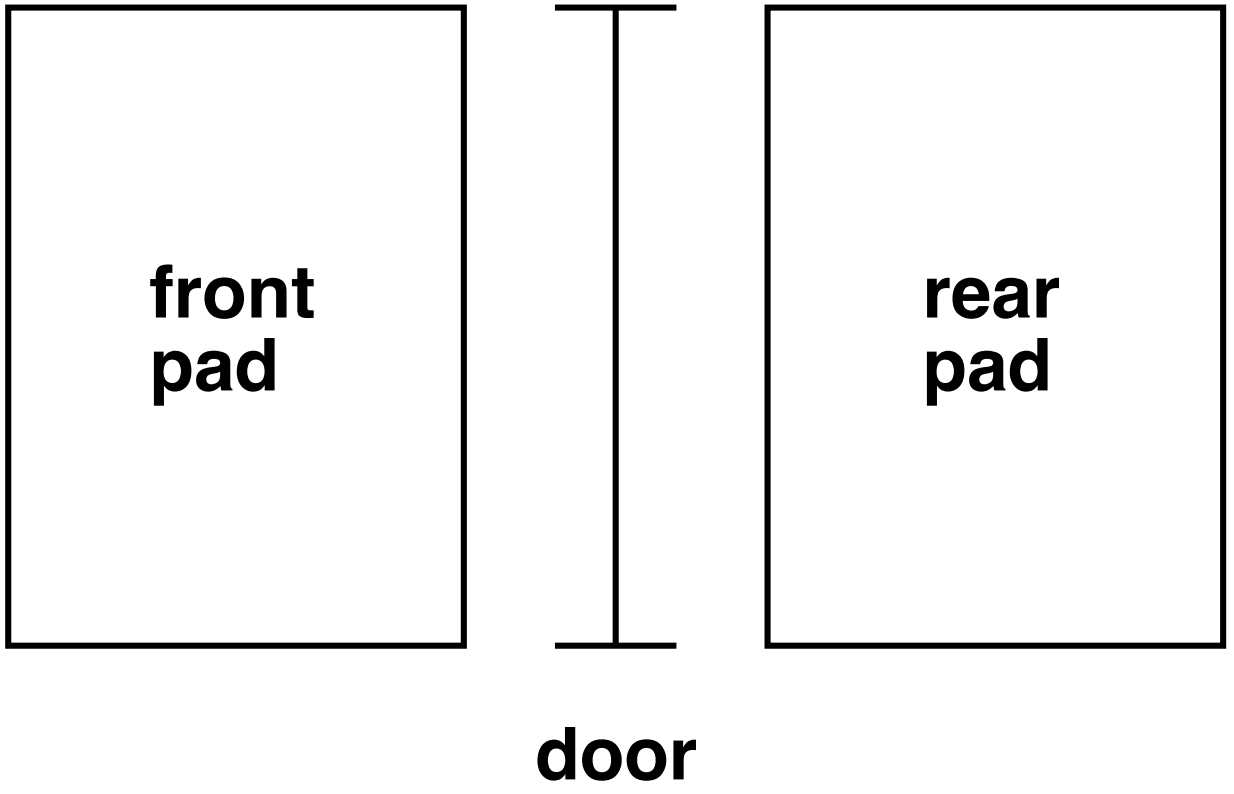
- formal definition of finite automata
- deterministic vs. non-deterministic finite automata
- regular languages
- operations on regular languages
- regular expressions

Example: A One-Way Automatic Door



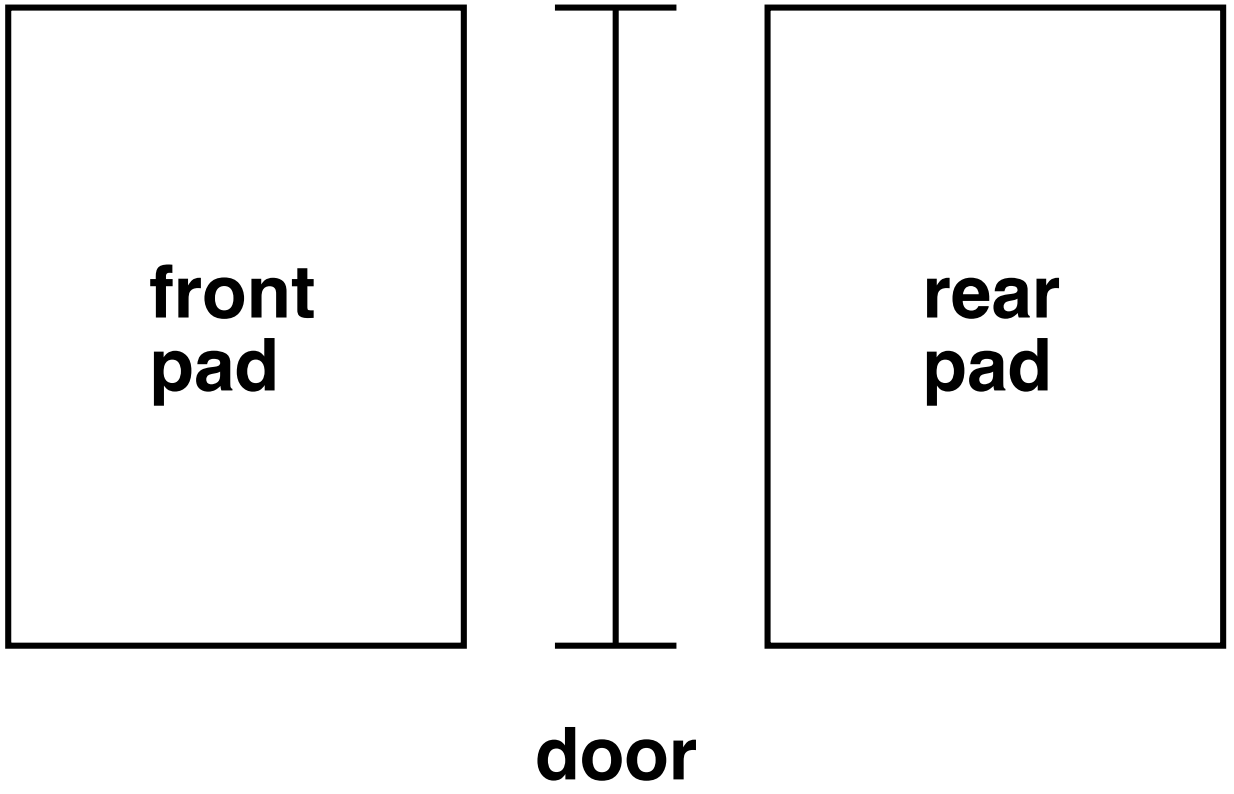
- open when person approaches

Example: A One-Way Automatic Door



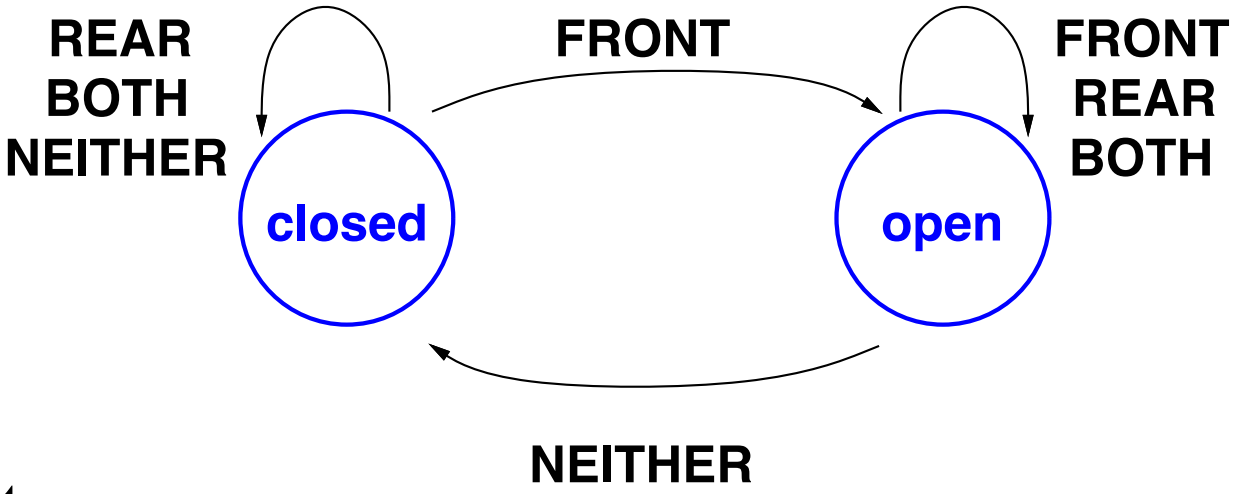
- open when person approaches
- hold open until person clears

Example: A One-Way Automatic Door



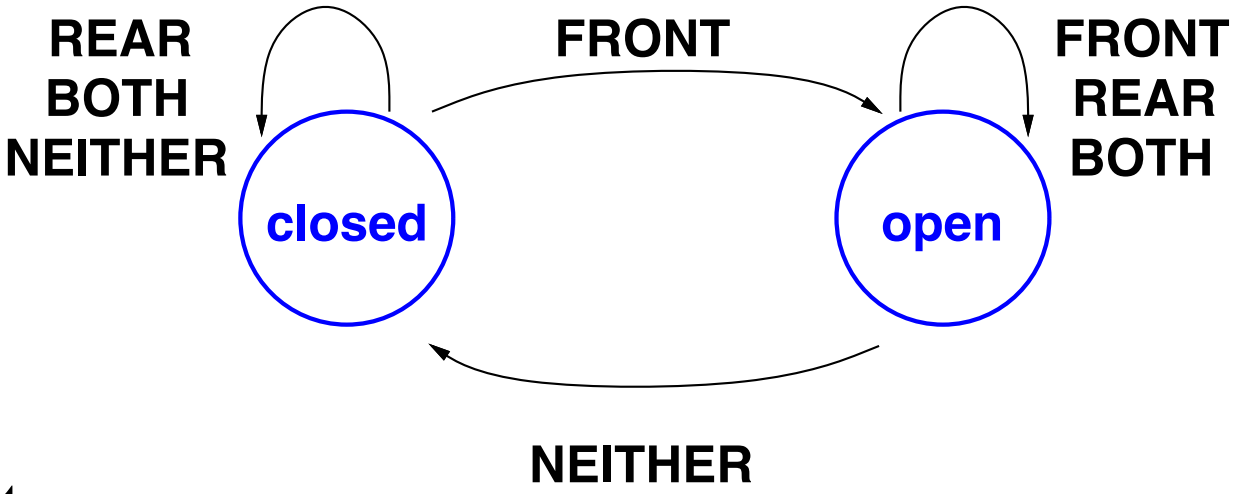
- open when person approaches
- hold open until person clears
- don’t open when someone standing behind door

The Automatic Door as DFA



- States:
  - OPEN
  - CLOSED

The Automatic Door as DFA



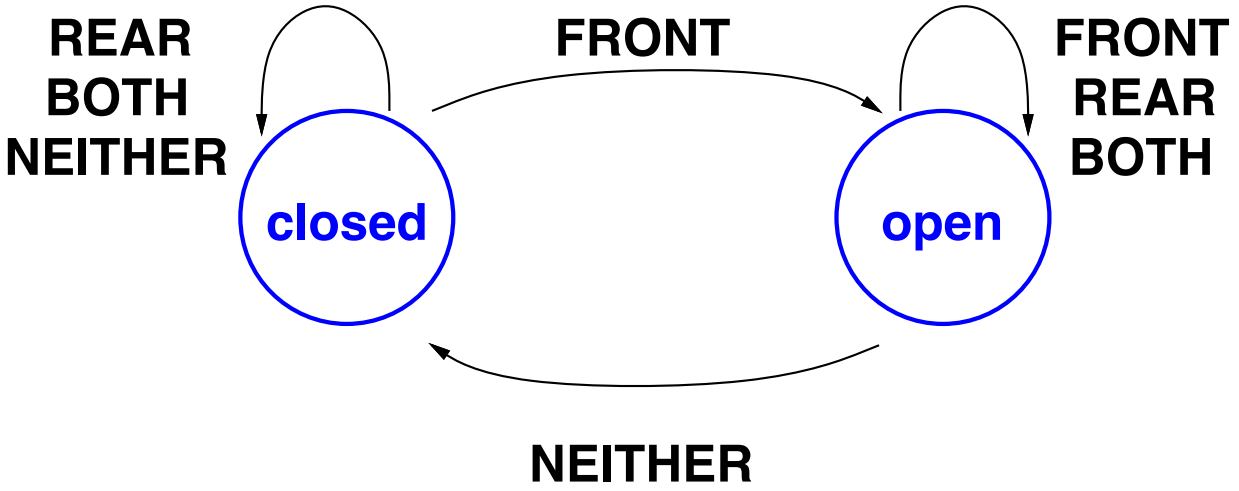
- States:
  - OPEN
  - CLOSED
- Sensor:
  - FRONT: someone on front pad
  - REAR: someone on rear pad
  - BOTH: someone(s) on both pads
  - NEITHER no one on either pad.

The Automatic Door as DFA

DFA is Deterministic Finite Automata

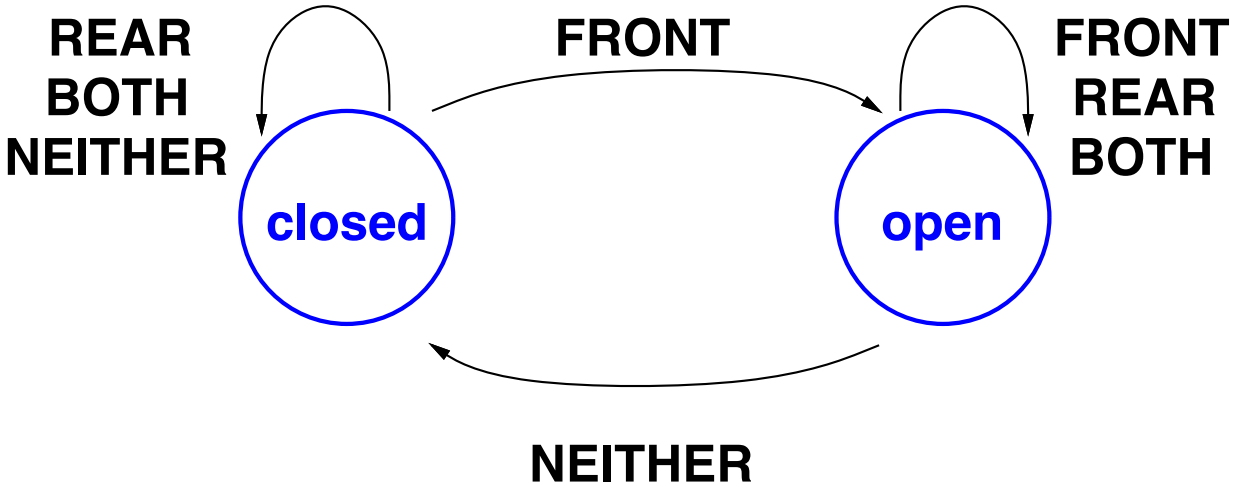
The Automatic Door as DFA

DFA is Deterministic Finite Automata



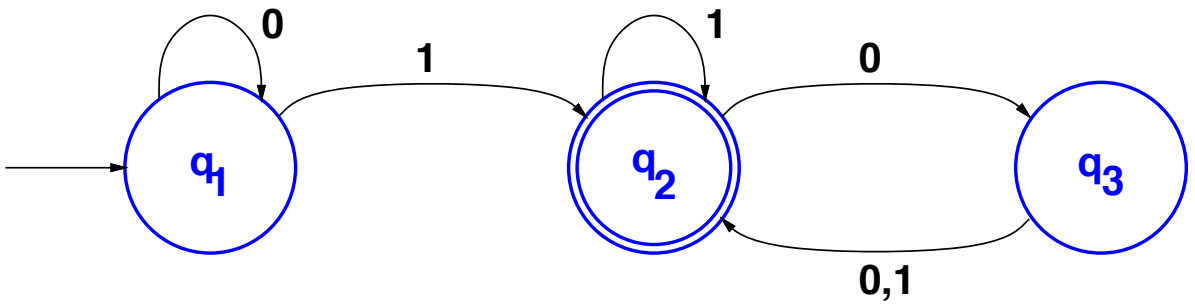
The Automatic Door as DFA

DFA is Deterministic Finite Automata



	neither	front	rear	both
closed	closed	open	closed	closed
open	closed	open	open	open

DFA: Informal Definition

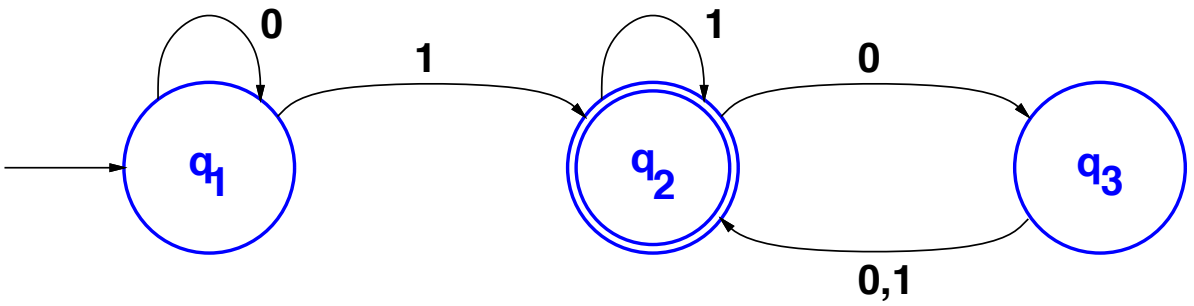


The machine

M_1:

- states: q_1, q_2, and q_3.

DFA: Informal Definition

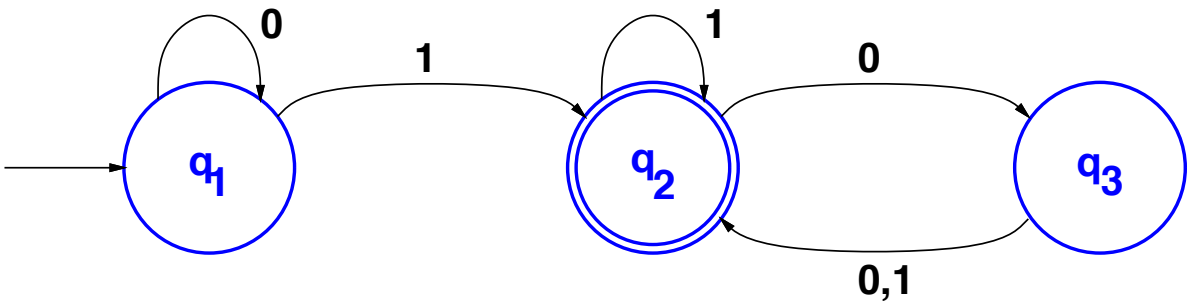


The machine

M_1:

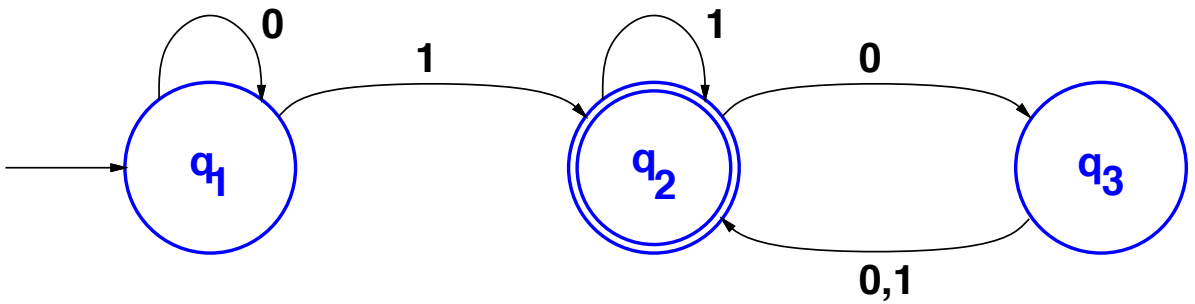
- states: q_1, q_2, and q_3.
- start state: q_1 (arrow from "outside").

DFA: Informal Definition (cont.)



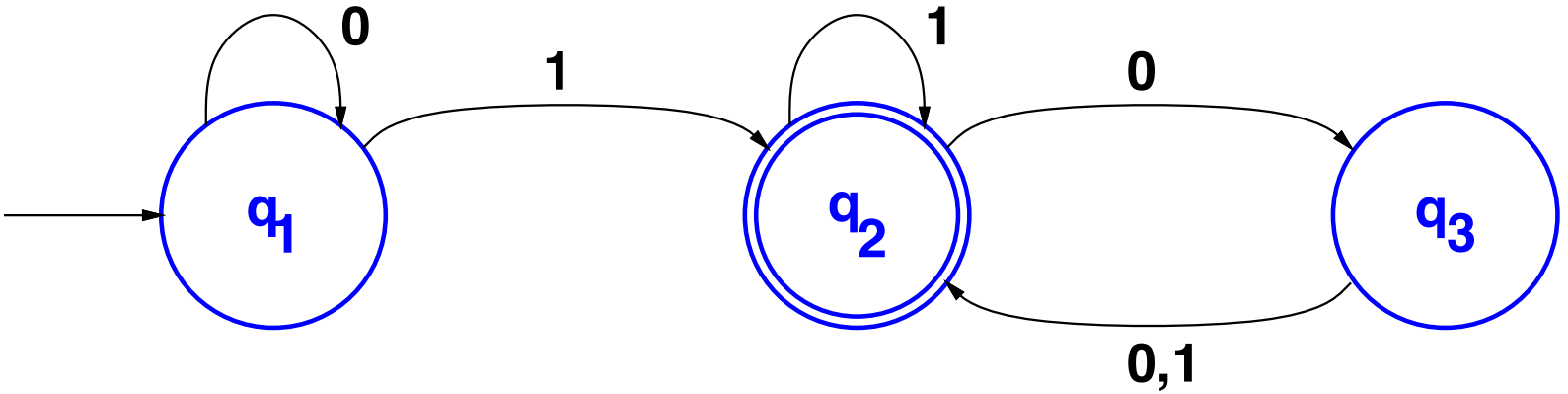
- On an input string
  - DFA begins in start state q_1
  - after reading each symbol, DFA makes state transition with matching label.

DFA: Informal Definition (cont.)



- On an input string
  - DFA begins in start state q_1
  - after reading each symbol, DFA makes state transition with matching label.
- After reading last symbol, DFA produces output:
  - accept if DFA is an accepting state.
  - reject otherwise.

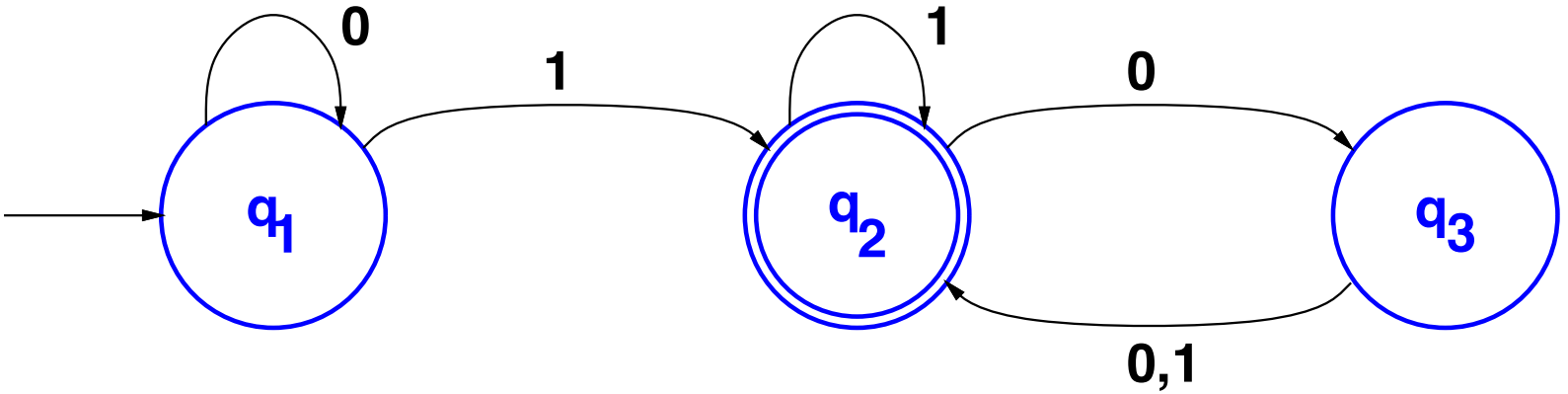
Informal Definition - Example



What happens on input strings

- 1101

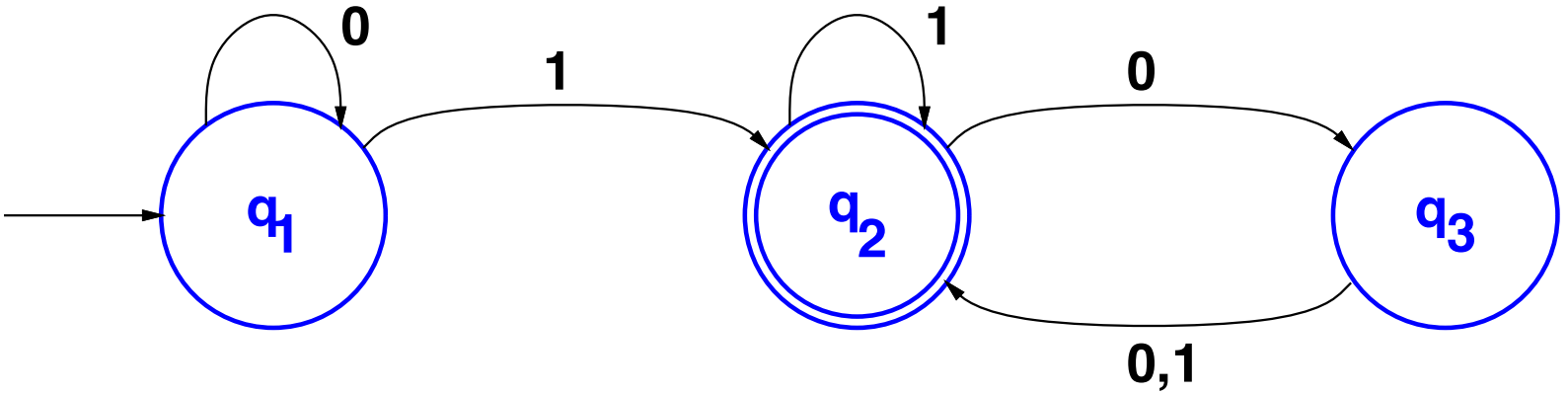
Informal Definition - Example



What happens on input strings

- 1101
- 0010

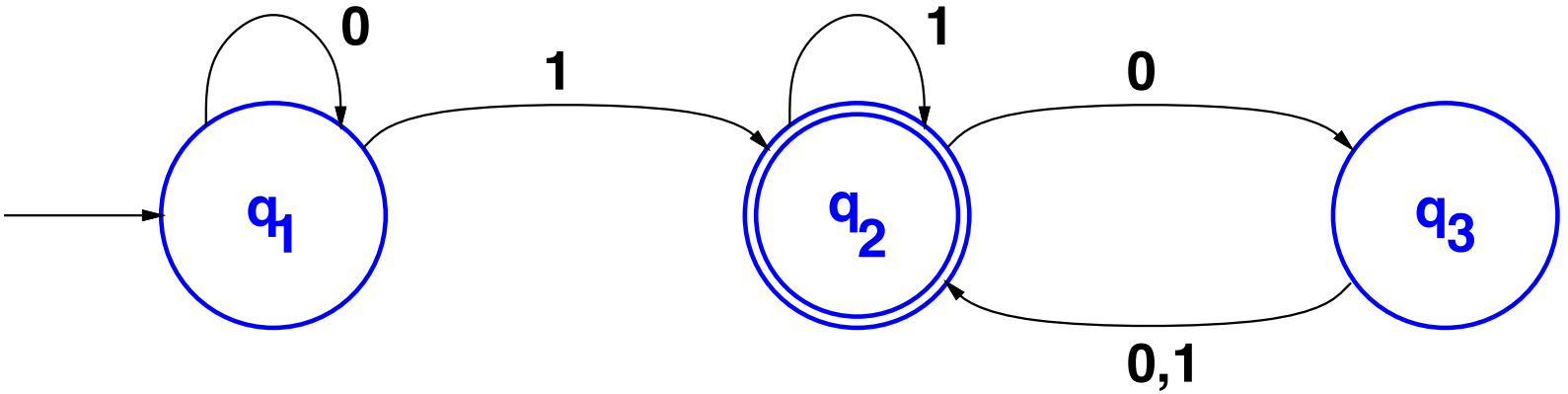
Informal Definition - Example



What happens on input strings

- 1101
- 0010
- 01100

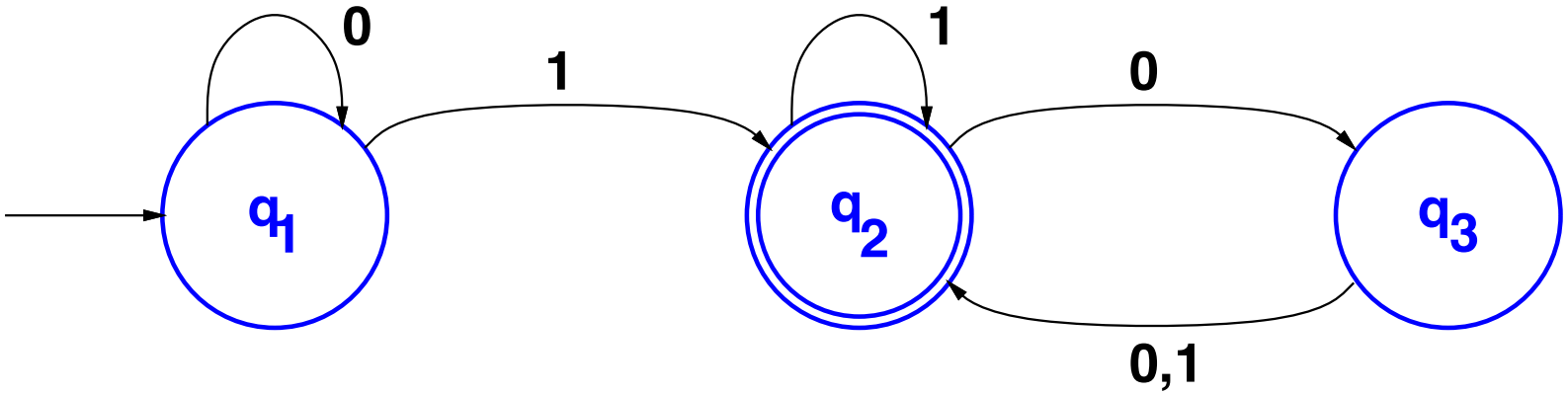
Informal Definition - Example



What happens on input strings

- 1101
- 0010
- 01100
- In general?!

Informal Definition



This DFA accepts

- all input strings that end with a 1
- all input strings that contain at least one 1, and end with an even number of 0's
- no other strings

Languages and Alphabets

An alphabet Σ is a finite set of letters.

- $\Sigma = \{a, b, c, \ldots, z\}$ – the English alphabet.
- $\Sigma = \{\alpha, \beta, \gamma, \ldots, \zeta\}$ – the Greek alphabet.
- $\Sigma = \{0, 1\}$ – the binary alphabet.
- $\Sigma = \{0, 1, \ldots, 9\}$ – the digital alphabet.

Languages and Alphabets

An alphabet Σ is a finite set of letters.

- $\Sigma = \{a, b, c, \ldots, z\}$ – the English alphabet.
- $\Sigma = \{\alpha, \beta, \gamma, \ldots, \zeta\}$ – the Greek alphabet.
- $\Sigma = \{0, 1\}$ – the binary alphabet.
- $\Sigma = \{0, 1, \ldots, 9\}$ – the digital alphabet.

The collection of all strings over Σ is denoted by Σ^*.

Languages and Alphabets

An alphabet Σ is a finite set of letters.

- $\Sigma = \{a, b, c, \ldots, z\}$ – the English alphabet.
- $\Sigma = \{\alpha, \beta, \gamma, \ldots, \zeta\}$ – the Greek alphabet.
- $\Sigma = \{0, 1\}$ – the binary alphabet.
- $\Sigma = \{0, 1, \ldots, 9\}$ – the digital alphabet.

The collection of all strings over Σ is denoted by Σ^*.

For the binary alphabet, $\varepsilon, 1, 0, 000000000,$ 1111111000 are all members of Σ^*.

Languages and Examples

A language over Σ is a subset $L \subseteq \Sigma^*$. For example

- Modern English.
- Ancient Greek.
- All prime numbers, written using digits.
- $A = \{w | w \text{ has at most seventeen 0's}\}$.
- $B = \{0^n 1^n | n \geq 0\}$.
- $C = \{w | w \text{ has an equal number of 0's and 1's}\}$.

Languages and DFA

Definition: $L(M)$, the language of a DFA M, is the set of strings L that M accepts, $L(M) = L$.

Languages and DFA

Definition: $L(M)$, the language of a DFA M, is the set of strings L that M accepts, $L(M) = L$.

Note that

- M may accept many strings, but
- M accepts only one language.

Languages and DFA

Definition: $L(M)$, the language of a DFA M, is the set of strings L that M accepts, $L(M) = L$.

Note that

- M may accept many strings, but
- M accepts only one language.

What language does M accept if it accepts no strings?

Languages and DFA

Definition: $L(M)$, the language of a DFA M, is the set of strings L that M accepts, $L(M) = L$.

Note that

- M may accept many strings, but
- M accepts only one language.

What language does M accept if it accepts no strings?

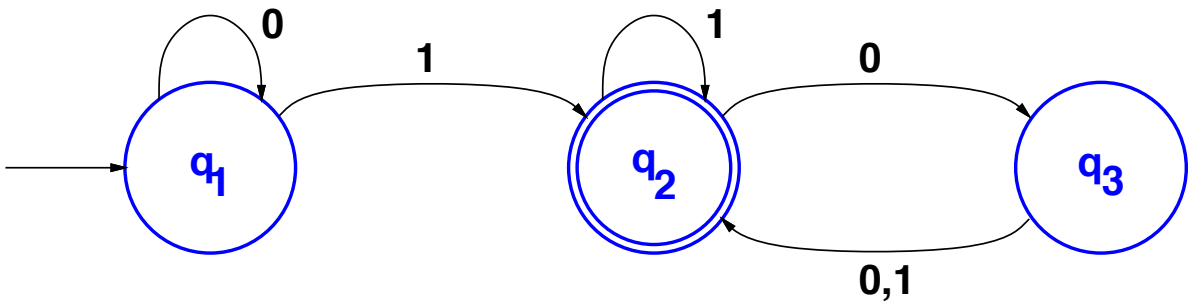
A language is called regular if some deterministic finite automaton accepts it.

Formal Definitions

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

- Q is a finite set called the states,
- Σ is a finite set called the alphabet,
- $\delta : Q \times \Sigma \to Q$ is the transition function,
- $q_0 \in Q$ is the start state, and
- $F \subseteq Q$ is the set of accept states.

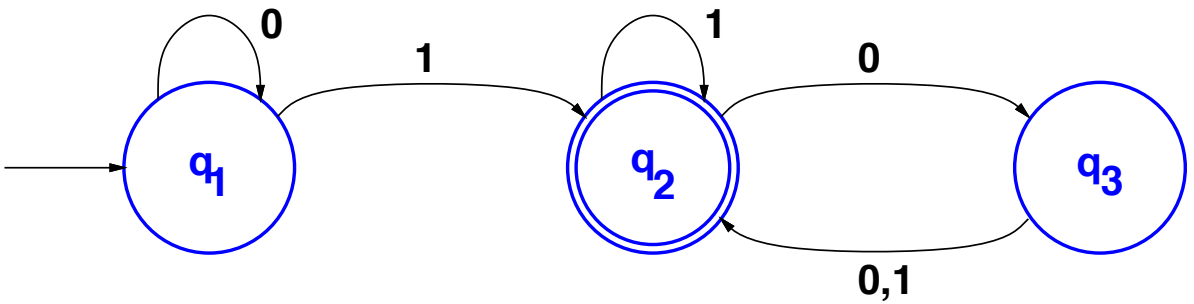
Back to M_1



$M_1 = (Q, \Sigma, \delta, q_1, F)$ where

- $Q = \{q_1, q_2, q_3\}, \Sigma = \{0, 1\},$

Back to M_1

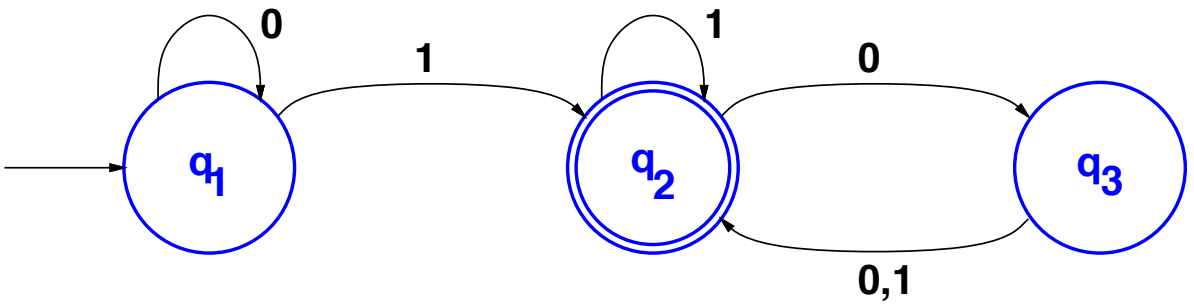


$M_1 = (Q, \Sigma, \delta, q_1, F)$ where

- $Q = \{q_1, q_2, q_3\}, \Sigma = \{0, 1\},$
- the transition function δ is

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

Back to M_1



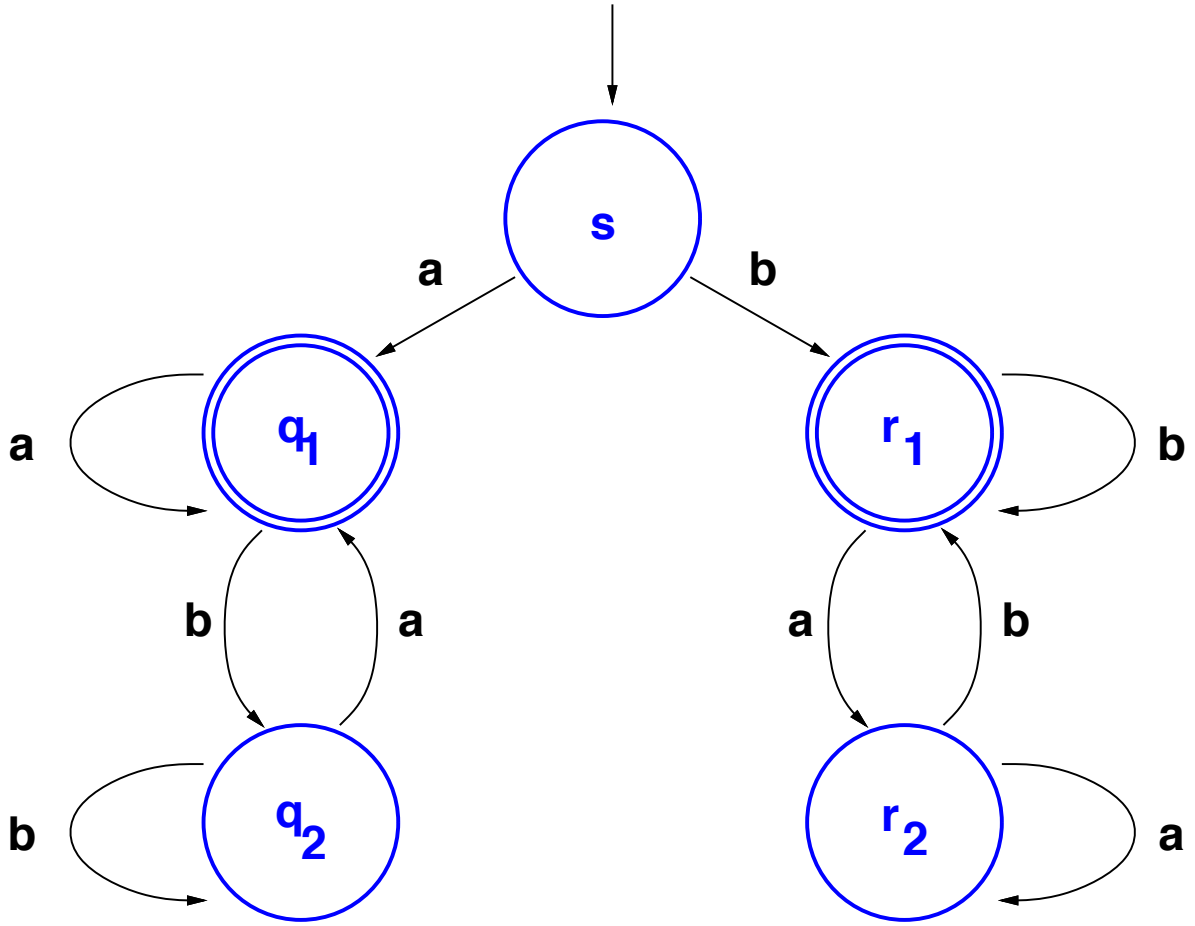
$M_1 = (Q, \Sigma, \delta, q_1, F)$ where

- $Q = \{q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$,
- the transition function δ is

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

- q_1 is the start state, and $F = \{q_2\}$.

Another Example



A Formal Model of Computation

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA, and
- let $w = w_1 w_2 \cdots w_n$ be a string over Σ.

We say that M accepts w if there is a sequence of states $r_0, \ldots, r_n$ ($r_i \in Q$) such that

- $r_0 = q_0$
- $\delta(r_i, w_{i+1}) = r_{i+1}, 0 \leq i < n$
- $r_n \in F$

The Regular Operations

Let A and B be languages.

The union operation:

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

The concatenation operation:

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

The star operation:

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

The Regular Operations – Examples

Let A= {good, bad} and B = {boy, girl}.

Union

$$A \cup B = \{\text{good}, \text{bad}, \text{boy}, \text{girl}\}$$

Concatenation

$$A \circ B = \{\text{goodboy}, \text{goodgirl}, \text{badboy}, \text{badgirl}\}$$

Star

$$A^* = \{\varepsilon, \text{good}, \text{bad}, \text{goodgood}, \text{goodbad}, \text{badbad}, \text{badgood}, \ldots\}$$

Claim: Closure Under Union

If A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

Claim: Closure Under Union

If A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

Approach to Proof:

- some M_1 accepts A_1
- some M_2 accepts A_2
- construct M that accepts $A_1 \cup A_2$.

Claim: Closure Under Union

If A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

Approach to Proof:

- some M_1 accepts A_1
- some M_2 accepts A_2
- construct M that accepts $A_1 \cup A_2$.

Attempted Proof Idea:

- first simulate M_1, and
- if M_1 doesn't accept, then simulate M_2.

Claim: Closure Under Union

If A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

Approach to Proof:

- some M_1 accepts A_1
- some M_2 accepts A_2
- construct M that accepts $A_1 \cup A_2$.

Attempted Proof Idea:

- first simulate M_1, and
- if M_1 doesn't accept, then simulate M_2.

What's wrong with this?

Claim: Closure Under Union

If A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

Approach to Proof:

- some M_1 accepts A_1
- some M_2 accepts A_2
- construct M that accepts $A_1 \cup A_2$.

Attempted Proof Idea:

- first simulate M_1, and
- if M_1 doesn't accept, then simulate M_2.

What's wrong with this?
Fix: Simulate both machines simultaneously.

Closure Under Union: Correct Proof

- Suppose $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accepts L_1,
- and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ accepts L_2.

Closure Under Union: Correct Proof

- Suppose $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accepts L_1,
- and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ accepts L_2.

Define M as follows (M will accept $L_1 \cup L_2$):

- $Q = Q_1 \times Q_2$.
- Σ is the same.

Closure Under Union: Correct Proof

- Suppose $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accepts L_1,
- and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ accepts L_2.

Define M as follows (M will accept $L_1 \cup L_2$):

- $Q = Q_1 \times Q_2$.
- Σ is the same.
- For each $(r_1, r_2) \in Q$ and $a \in \Sigma$,
  $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$

Closure Under Union: Correct Proof

- Suppose $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accepts L_1,
- and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ accepts L_2.

Define M as follows (M will accept $L_1 \cup L_2$):

- $Q = Q_1 \times Q_2$.
- Σ is the same.
- For each $(r_1, r_2) \in Q$ and $a \in \Sigma$,
  $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
- $q_0 = (q_1, q_2)$

Closure Under Union: Correct Proof

- Suppose $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accepts L_1,
- and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ accepts L_2.

Define M as follows (M will accept $L_1 \cup L_2$):

- $Q = Q_1 \times Q_2$.
- Σ is the same.
- For each $(r_1, r_2) \in Q$ and $a \in \Sigma$,
  $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
- $q_0 = (q_1, q_2)$
- $F = \{(r_1, r_2) | r_1 \in F_1 \text{ or } r_2 \in F_2\}$. ♣