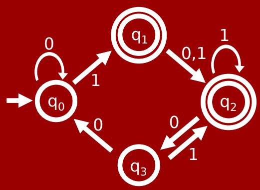
15-251: Great Theoretical Ideas in Computer Science
Lecture 4

Finite Automata

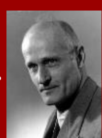


What is **computation**?
What is an **algorithm**?

How can we mathematically define them?

Inspirational quotation #1:

"An algorithm is a finite answer to an infinite number of questions"



Stephen Kleene

Inspirational quotation #2:

An algorithm solves a problem if it gives the correct solution on every instance.

We'll define last 3 terms now.
We'll save algorithm for later.

me

What is a computational **problem**?

We'll start with some examples.

Example problem 1:

PRIMALITY

Instance (*also known as **input***)	***Solution***
0	No
1	No
2	Yes
3	Yes
4	No
•••	•••
42	No
•••	•••
251	Yes
•••	•••
170141183460469231731687303715884105727	Yes

Example problem 2:

PALINDROME

Instance (also known as ***input***)	*Solution*
`a`	`Yes`
`10101`	`Yes`
`selfless`	`No`
`dammitimmad`	`Yes`
`zxckallkdsflsdkf`	`No`
`parahazıramarızaharap`	`Yes`

These are examples of decision problems:
Problems where the solution is **Yes / No**.

(Also known as **True / False**, **1 / 0**, **accept / reject**.)

Example problem 3:

MULTIPLICATION

Instance	*Solution*
`3, 7`	`21`
`610, 25`	`15250`
`50, 610`	`30500`
`15251, 252`	`3843252`
`12345679, 9`	`111111111`

Example problem 4:

SORTING

Instance

`[vanilla, mind, Anil, yogurt, doesn’t]`

Solution

`[Anil, doesn’t, mind, vanilla, yogurt]`

A problem is a collection of (naturally related) instances and solutions.

Representing problems

The instances of a problem can be:

- numbers
- strings
- pairs of numbers
- lists of strings
- trees
- graphs
- images
- …

As you know…

Can all be conveniently encoded by **strings**.
Even just by **binary** (`0`/`1`) strings.

String notation

Alphabet: A nonempty finite set Σ of symbols.
$\Sigma = \{0,1\}$ is a popular choice.

String: A finite sequence of 0 or more symbols.
(or "word") The length-0 string is denoted ε.
Σ^n means all strings over Σ of length n.
Σ^* means **all** strings over Σ.

Language: A collection of strings.
In other words any subset $L \subseteq \Sigma^*$.

Representing problems

We can encode instances/solutions with strings.

Thus we can think of a **problem** as a **function**

$$f : \Sigma^* \to \Sigma^*$$

mapping instances to solutions.

A **decision problem** can be thought of as

$$f : \Sigma^* \to \{\text{No}, \text{Yes}\}$$

Representing problems

A **decision problem** can be thought of as

$$f : \Sigma^* \to \{\text{No}, \text{Yes}\}$$

or equivalently as a **language**

$$L \subseteq \Sigma^*$$

$L = \{x \in \Sigma^* : f(x) = \text{Yes}\}$ $\qquad f(x) = \begin{cases} \text{Yes} & \text{if } x \in L \\ \text{No} & \text{if } x \notin L \end{cases}$

E.g.: PALINDROME $= \{x \in \Sigma^* : x = \text{Reverse}(x)\}$

What is **computation**? What is an **algorithm**?

This lecture:
A very simple computational model:
Deterministic Finite Automata

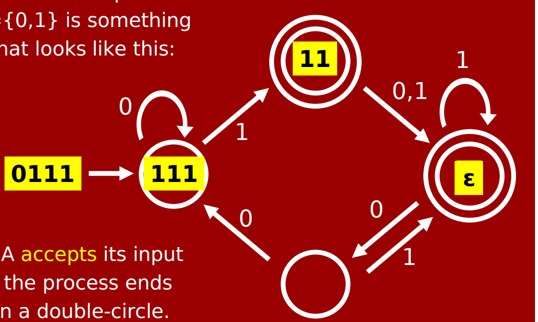
It's so wimpy, it can only implement an extremely restricted kind of algorithm.

Has some interesting applications.

A good warmup before we study general models of computations next lecture.

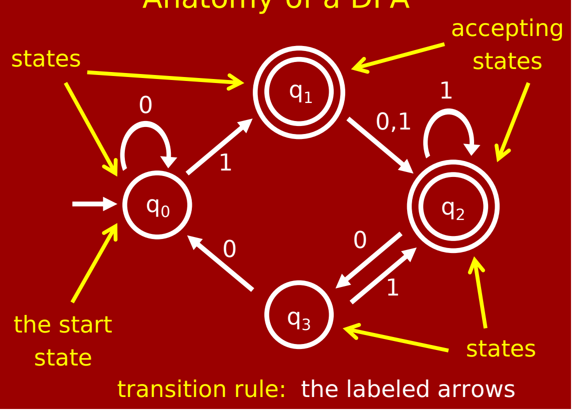
Deterministic Finite Automata (DFAs)

A DFA over alphabet $\Sigma=\{0,1\}$ is something that looks like this:



DFA accepts its input if the process ends in a double-circle.

Anatomy of a DFA



Computing with DFAs

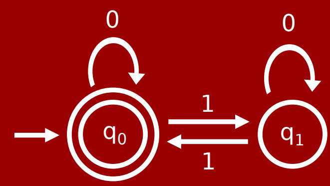
Let M be a DFA, using alphabet Σ.

We think of M as a computing mechanism, which "accepts" some strings in Σ^* and "rejects" the others.

Definition: $L(M) = \{x \in \Sigma^* : M \text{ accepts } x\}$

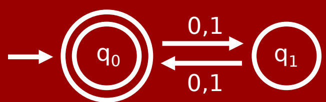
Called the "language decided/accepted by M".

If P is a decision problem, we say M solves it if L(M) = P.



What language does this DFA decide?

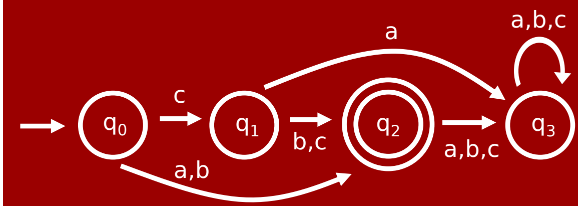
All binary strings with an even number of 1's.



What language does this DFA decide?

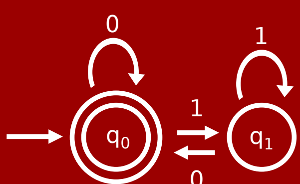
All binary strings with even length.

M is the following DFA, with alphabet $\Sigma=\{a,b,c\}$:



$L(M) = \{a, b, cb, cc\} \subseteq \{a,b,c\}^*$

M:



$L(M) = \{x : x \text{ ends in a } 0\} \cup \{\varepsilon\}$

Formal definition of DFAs

Let's give a very formal definition of DFAs.

Having some notations can help us reason about them.

Also illustrates that we can completely formalize this notion of computation.

Formal definition of DFAs

A deterministic finite automaton is a 5-tuple:

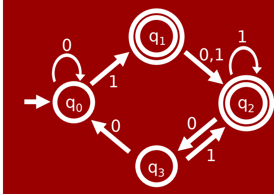
$M = (Q, \Sigma, \delta, q_0, F)$

where Q is a nonempty finite set of states,
Σ is an alphabet,
$\delta : Q \times \Sigma \rightarrow Q$ is the transition function,
$q_0 \in Q$ is the start state,
$F \subseteq Q$ is the set of accepting states.

Formal definition of DFAs

A deterministic finite automaton is a 5-tuple:

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



$Q = \{q_0, q_1, q_2, q_3\}$

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

δ we'll come back to

q_0 is the start state

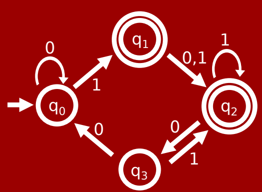
$F = \{q_1,q_2\}$

Formal definition of DFAs

A deterministic finite automaton is a 5-tuple:

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

$\delta : Q \times \{0,1\} \rightarrow Q$ is…



δ	0	1
q_0	q_0	q_1
q_1	q_2	q_2
q_2	q_3	q_2
q_3	q_0	q_2

Formal definition of DFAs

Let $w = w_1 w_2 w_3 \cdots w_n$, where each $w_i \in \Sigma$.

We say that M **accepts** string w if:

There exist states $r_0, r_1, r_2, , \ldots, r_n \in Q$ such that:

- $r_0 = q_0$, the initial state;
- $\delta(r_{t-1}, w_t) = r_t$ for all $t = 1, 2, 3, \ldots, n$;
- $r_n \in F$ (the accepting states).

Otherwise we say M **rejects** w.

The sequence $r_0, r_1, r_2, , \ldots, r_n$ is called the computation trace.

DFA-construction practice:

$U = \{110, 101\}$

$U^c = \{0,1\}^* \setminus \{110, 101\}$

$P = \{x \in \{0,1\}^* : x$ starts and ends with same bit$\}$

$D = \{x \in \{0,1\}^* : |x|$ divisible by 2 or by 3$\}$

$S = \{\varepsilon, 110, 110110, 110110110, 110110110110, \ldots\}$

$G = \{x \in \{0,1\}^* : x$ contains the substring 110$\}$

$C = \{x \in \{0,1\}^* :$ 10 and 01 occur equally often in $x\}$

Regular Languages

Definition:

A language $L \subseteq \Sigma^*$ is **regular** if there is a DFA which decides it.

Questions:

1. Are all languages regular?
2. Are there other ways to tell if L is regular?

Theorem: $L = \{0^n1^n : n \in \mathbb{N}\}$ is not regular

Notation:

If $s \in \Sigma$ is a symbol and $n \in \mathbb{N}$ then s^n denotes the string $sss\cdots s$ (n times).

E.g., s^3 means sss, s^5 means $sssss$,
s^1 means s, s^0 means ε, etc.

Thus $L = \{\varepsilon, 01, 0011, 000111, 00001111, \ldots\}$.

Theorem: $L = \{0^n1^n : n \in \mathbb{N}\}$ is not regular

Intuition:

For a DFA to decide L, it *seems* like it needs to "remember" how many 0's it sees at the beginning of the string, so that it can "check" there are equally many 1's.

But a DFA has only finitely many states — shouldn't be able to handle arbitrary n.

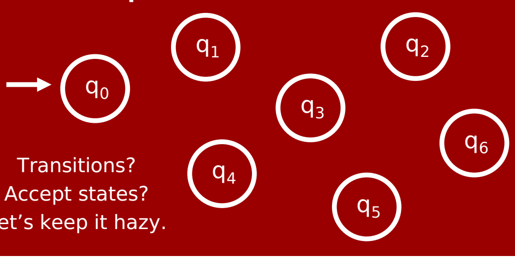
But we need to be careful: the following language **is** regular:

$C = \{x \in \{0,1\}^* : 10 \text{ and } 01 \text{ occur equally often in } x\}$

Theorem: $L = \{0^n1^n : n \in \mathbb{N}\}$ is not regular

Warmup: No DFA with, say, 7 states can decide L.

Input: 00000000000001111111111111

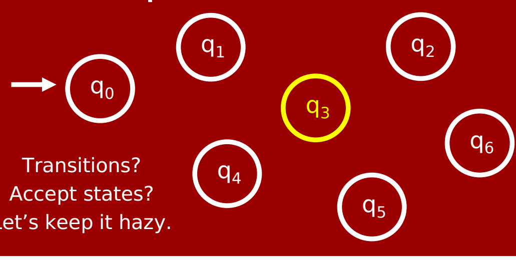


Transitions?
Accept states?
Let's keep it hazy.

Theorem: $L = \{0^n1^n : n \in \mathbb{N}\}$ is not regular

Warmup: No DFA with, say, 7 states can decide L.

Input: 00000000000001111111111111

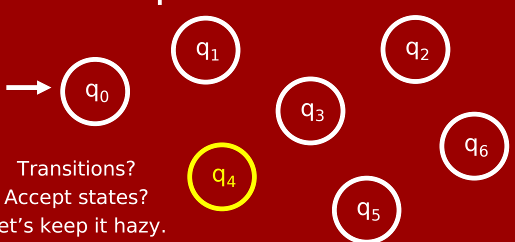


Transitions?
Accept states?
Let's keep it hazy.

Theorem: $L = \{0^n1^n : n \in \mathbb{N}\}$ is not regular

Warmup: No DFA with, say, 7 states can decide L.

Input: 00000000000001111111111111

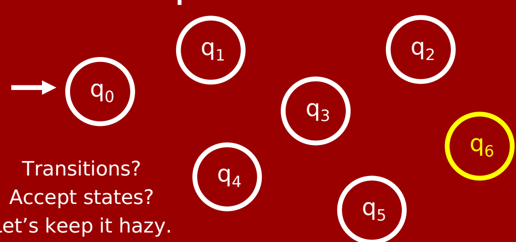


Transitions?
Accept states?
Let's keep it hazy.

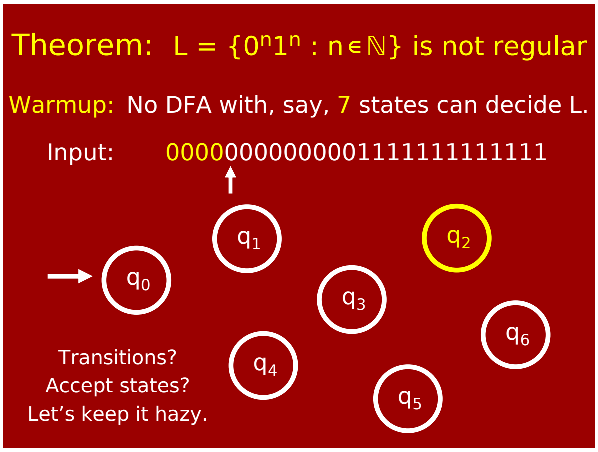
Theorem: $L = \{0^n1^n : n \in \mathbb{N}\}$ is not regular

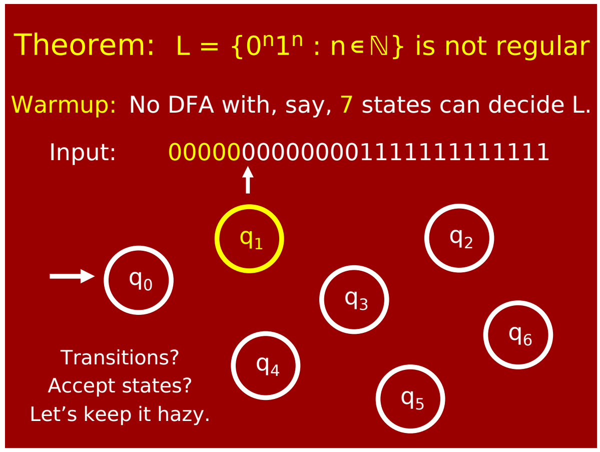
Warmup: No DFA with, say, 7 states can decide L.

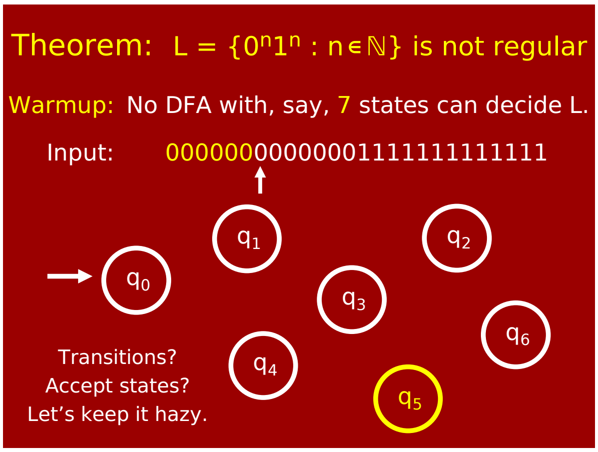
Input: 00000000000001111111111111

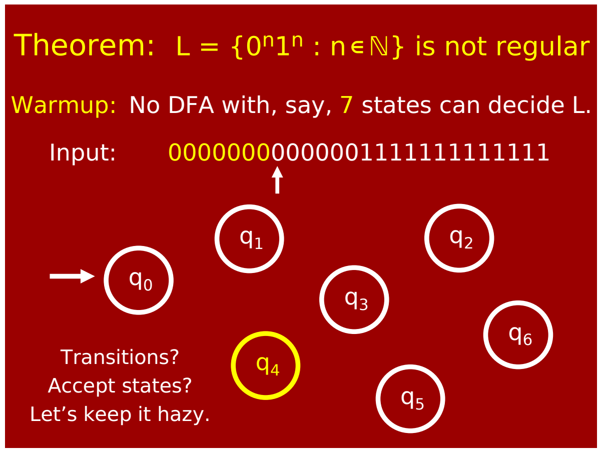


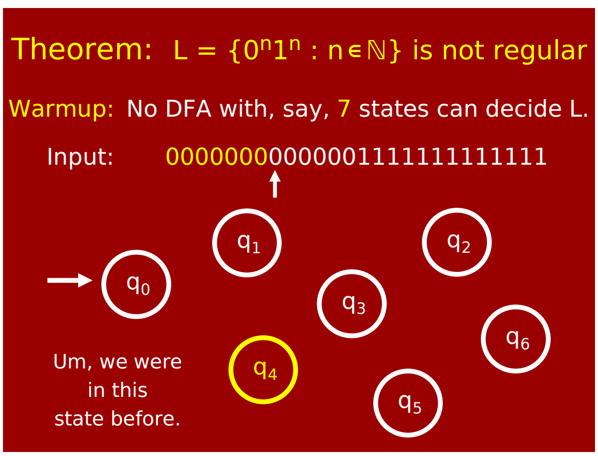
Transitions?
Accept states?
Let's keep it hazy.

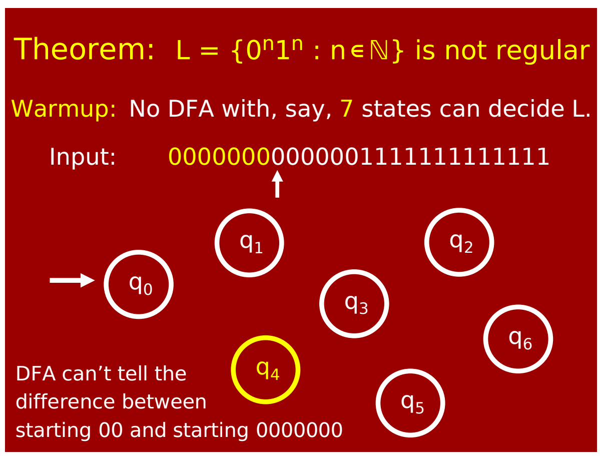

Theorem: L = {0^n1^n : n ∈ ℕ} is not regular
Warmup: No DFA with, say, 7 states can decide L.
Input: 00000000000001111111111111
q0
q1
q2
q3
q4
q5
q6
Transitions?
Accept states?
Let’s keep it hazy.


Theorem: L = {0^n1^n : n ∈ ℕ} is not regular
Warmup: No DFA with, say, 7 states can decide L.
Input: 00000000000001111111111111
q0
q1
q2
q3
q4
q5
q6
Transitions?
Accept states?
Let’s keep it hazy.


Theorem: L = {0^n1^n : n ∈ ℕ} is not regular
Warmup: No DFA with, say, 7 states can decide L.
Input: 00000000000001111111111111
q0
q1
q2
q3
q4
q5
q6
Transitions?
Accept states?
Let’s keep it hazy.


Theorem: L = {0^n1^n : n ∈ ℕ} is not regular
Warmup: No DFA with, say, 7 states can decide L.
Input: 00000000000001111111111111
q0
q1
q2
q3
q4
q5
q6
Transitions?
Accept states?
Let’s keep it hazy.

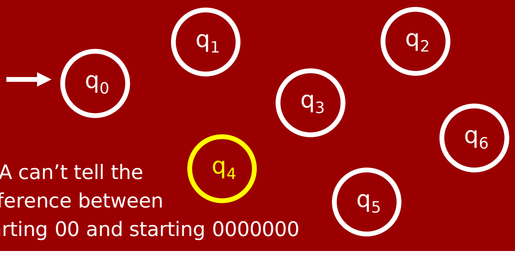

Theorem: L = {0^n1^n : n ∈ ℕ} is not regular
Warmup: No DFA with, say, 7 states can decide L.
Input: 00000000000001111111111111
q0
q1
q2
q3
q4
q5
q6
Um, we were
in this
state before.


Theorem: L = {0^n1^n : n ∈ ℕ} is not regular
Warmup: No DFA with, say, 7 states can decide L.
Input: 00000000000001111111111111
q0
q1
q2
q3
q4
q5
q6
DFA can’t tell the
difference between
starting 00 and starting 0000000

Theorem: $L = \{0^n 1^n : n \in \mathbb{N}\}$ is not regular

Warmup: No DFA with, say, 7 states can decide L.

Input: 0000000 → If the rest of the input is actually 1111111, the DFA better accept

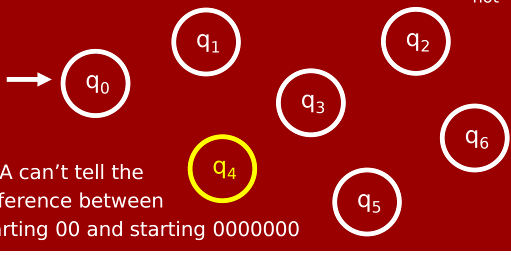


DFA can't tell the difference between starting 00 and starting 0000000

Theorem: $L = \{0^n 1^n : n \in \mathbb{N}\}$ is not regular

Warmup: No DFA with, say, 7 states can decide L.

Input: 00 → If the rest of the input is actually 1111111, the DFA better not accept



DFA can't tell the difference between starting 00 and starting 0000000

Theorem: $L = \{0^n 1^n : n \in \mathbb{N}\}$ is not regular

Full proof:

Suppose for contradiction DFA M decides L using, say, k states.

Let r_i denote the state M reaches after processing 0^i.

By Pigeonhole, there is a repeat among $r_0, r_1, r_2, \ldots, r_k$.

So say that $r_s = r_t$ for some $0 \le s \ne t \le k$.

Since $0^s 1^s \in L$, starting from r_s and processing 1^s causes M to reach an accepting state.

So on input $0^t 1^s$, M will process 0^t, reach state $r_t = r_s$, process 1^s, and therefore reach an accepting state.

But $0^t 1^s \notin L$ since $s \ne t$, a contradiction. □

Proving a language L is not regular

Most of the time, the proof looks like this:

1. Assume for contradiction there is a DFA M which decides language L.
2. Argue (usually by Pigeonhole) there are two strings x and y which reach the same state in M.
3. Show there is a string z such that $xz \in L$ but $yz \notin L$. Contradiction, since M acts the same (accept/reject) on both.

Regular Languages

Definition:

A language $L \subseteq \Sigma^*$ is **regular** if there is a DFA which decides it.

Questions:

1. Are all languages regular?
2. Are there other ways to tell if L is regular?

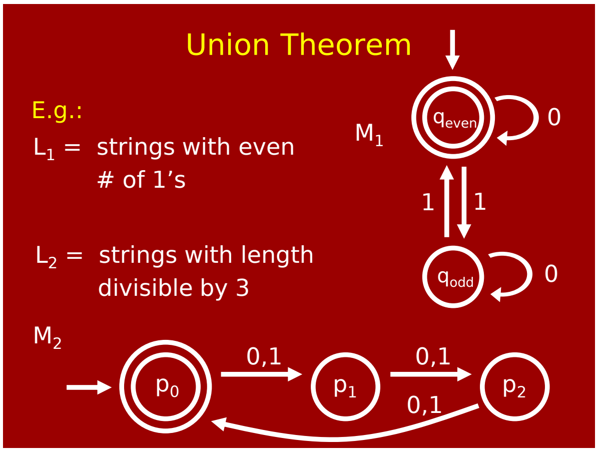
Union Theorem

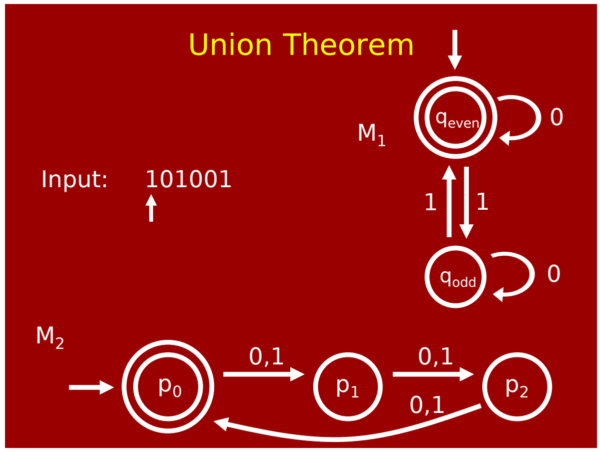
Definition:

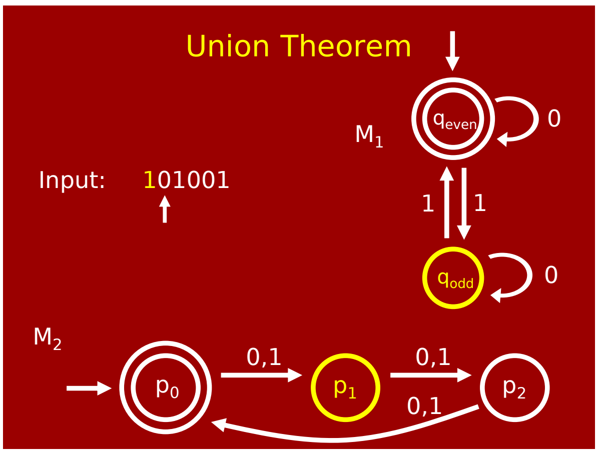
Let L_1 and L_2 be any languages.
Their union, $L_1 \cup L_2$, is $\{x : x \in L_1 \text{ or } x \in L_2\}$.

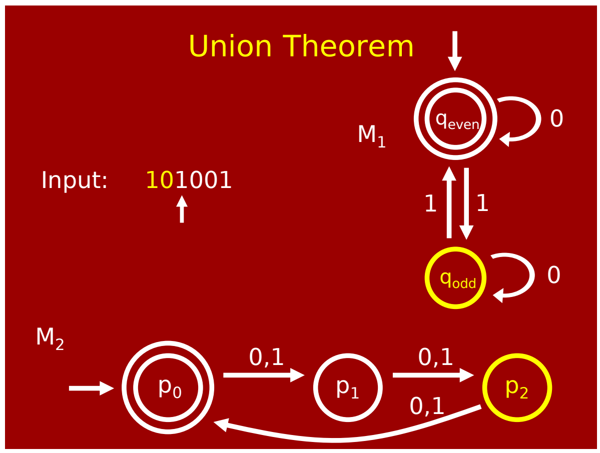
Union Theorem:

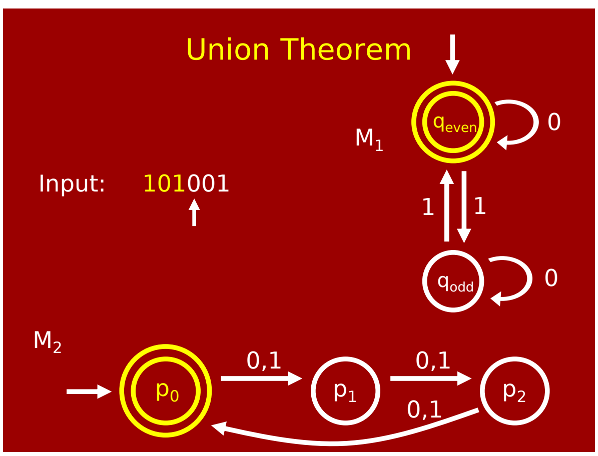
If L_1 and L_2 are both regular languages over Σ then so is $L_1 \cup L_2$.

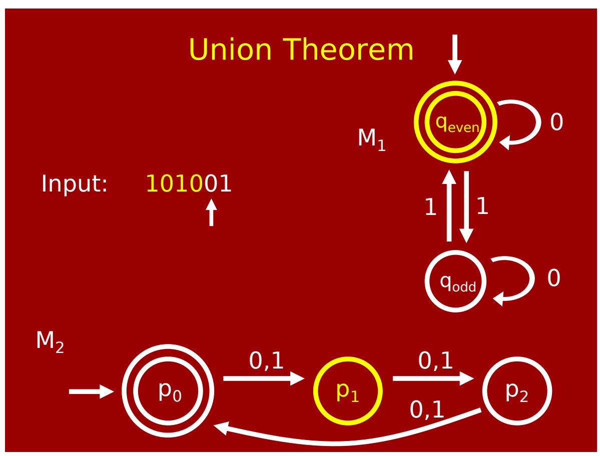

Union Theorem
E.g.:
L1 = strings with even # of 1's
L2 = strings with length divisible by 3
M1
qeven
qodd
0
1
1
0
M2
p0
p1
p2
0,1
0,1
0,1

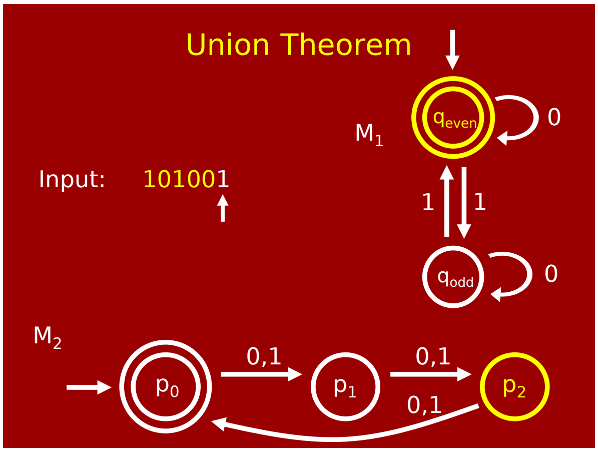

Union Theorem
Input: 101001
M1
qeven
qodd
0
1
1
0
M2
p0
p1
p2
0,1
0,1
0,1

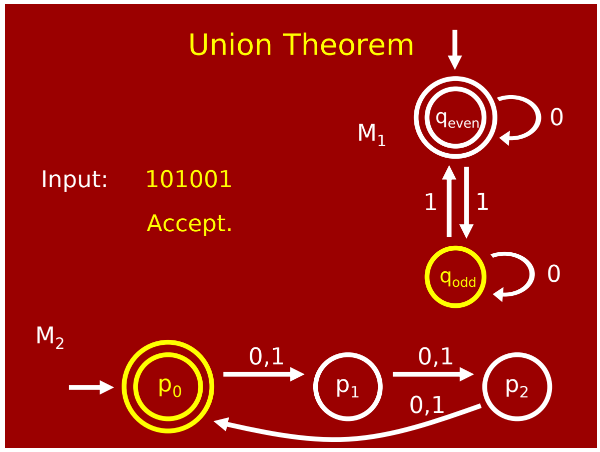

Union Theorem
Input: 101001
M1
qeven
qodd
0
1
1
0
M2
p0
p1
p2
0,1
0,1
0,1

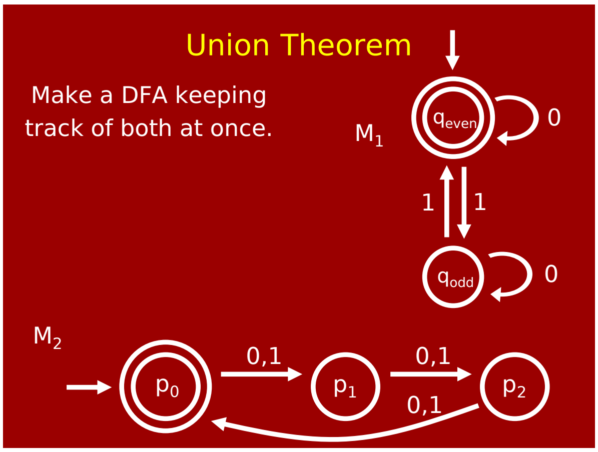

Union Theorem
Input: 101001
M1
qeven
qodd
0
1
1
0
M2
p0
p1
p2
0,1
0,1
0,1

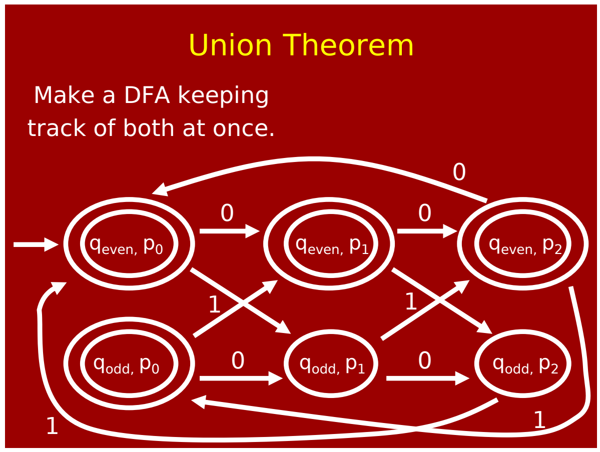

Union Theorem
Input: 101001
M1
qeven
qodd
0
1
1
0
M2
p0
p1
p2
0,1
0,1
0,1

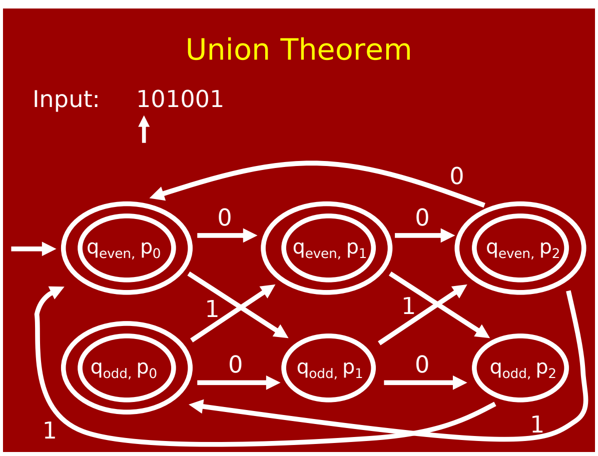

Union Theorem
Input: 101001
M1
qeven
qodd
0
1
1
0
M2
p0
p1
p2
0,1
0,1
0,1

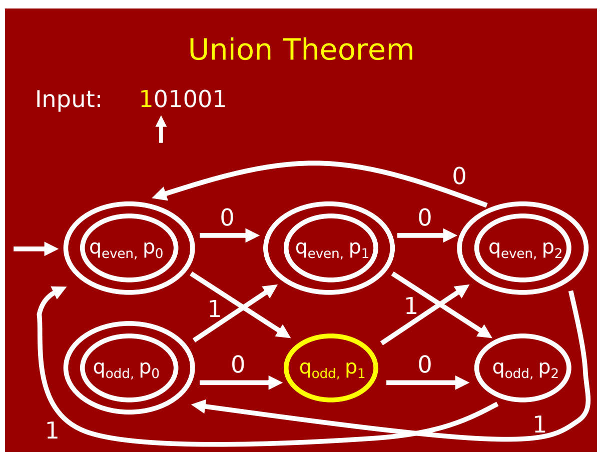

Union Theorem
Input: 101001
M_1
q_{even}
q_{odd}
0
1
1
0
M_2
p_0
p_1
p_2
0,1
0,1
0,1


Union Theorem
Input: 101001
Accept.
M_1
q_{even}
q_{odd}
0
1
1
0
M_2
p_0
p_1
p_2
0,1
0,1
0,1


Union Theorem
Make a DFA keeping track of both at once.
M_1
q_{even}
q_{odd}
0
1
1
0
M_2
p_0
p_1
p_2
0,1
0,1
0,1

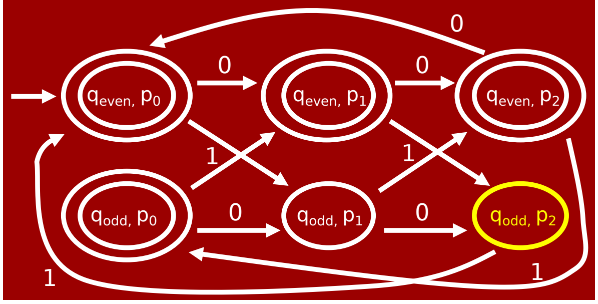

Union Theorem
Make a DFA keeping track of both at once.
q_{even}, p_0
q_{even}, p_1
q_{even}, p_2
q_{odd}, p_0
q_{odd}, p_1
q_{odd}, p_2
0
0
0
1
1
0
0
1
1


Union Theorem
Input: 101001
q_{even}, p_0
q_{even}, p_1
q_{even}, p_2
q_{odd}, p_0
q_{odd}, p_1
q_{odd}, p_2
0
0
0
1
1
0
0
1
1


Union Theorem
Input: 101001
q_{even}, p_0
q_{even}, p_1
q_{even}, p_2
q_{odd}, p_0
q_{odd}, p_1
q_{odd}, p_2
0
0
0
1
1
0
0
1
1

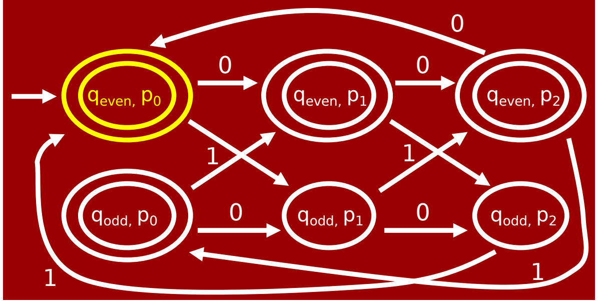
Union Theorem

Input: 101001



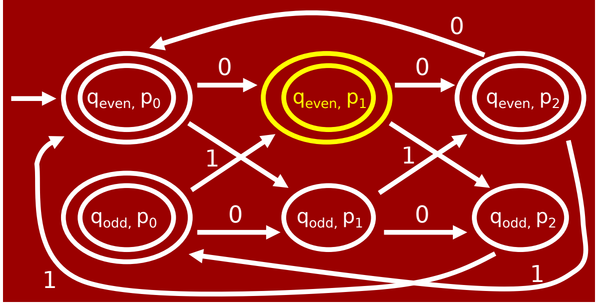
Union Theorem

Input: 101001



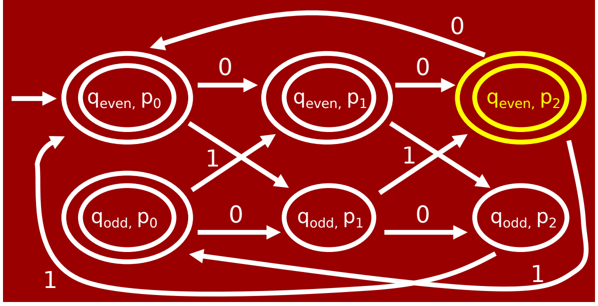
Union Theorem

Input: 101001



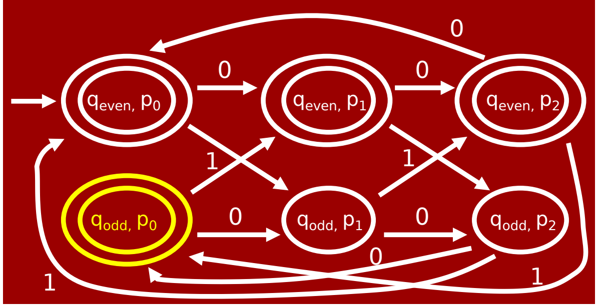
Union Theorem

Input: 101001



Union Theorem

Input: 101001
Accept.



Union Theorem

Formal proof:

Suppose L_1 is decided by $M_1 = (Q, \Sigma, \delta, q_0, F)$.

Suppose L_2 is decided by $M_2 = (Q', \Sigma, \delta', q_0', F')$.

Define the DFA $M = (Q \times Q', \Sigma, \beta, (q_0, q_0'), G)$,

where $G = \{(q,q') : q \in F$ **or** $q' \in F'\}$

and $\beta((q,q'), c) = (\delta(q,c), \delta'(q',c))$.

Then…(it's not hard to see that)… $L(M) = L_1 \cup L_2$.

More “closure” theorems

Theorem: $L_1 \cup L_2$ is regular if L_1, L_2 are.

“Concatenation”: $L_1 \cdot L_2 = \{xy : x \in L_1, y \in L_2\}$

Theorem: $L_1 \cdot L_2$ is regular if L_1, L_2 are.

“Star”: $L^* = \{ x_1 x_2 \cdots x_k : k \geq 0, \text{each } x_i \text{ in } L\}$

Theorem: L^* is regular if L is.

The Regular Operations

Theorem: $L_1 \cup L_2$ is regular if L_1, L_2 are.

Theorem: $L_1 \cdot L_2$ is regular if L_1, L_2 are.

Theorem: L^* is regular if L is.

The latter two theorems are somewhat more tricky to prove.

You will prove them on the homework!

A Deductive System for regular languages

Objects: *Languages* over alphabet Σ

Initial objects: $\varnothing$, $\{a\}$ for each $a \in \Sigma$

Deduction rules:

From L_1, L_2, can deduce $L_1 \cup L_2$
From L_1, L_2, can deduce $L_1 \cdot L_2$
From L, can deduce L^*

From the previous slide, we know that any deducible language is regular.

A Deductive System for regular languages

Objects: *Languages* over alphabet Σ

Initial objects: $\varnothing$, $\{a\}$ for each $a \in \Sigma$

Deduction rules:

From L_1, L_2, can deduce $L_1 \cup L_2$
From L_1, L_2, can deduce $L_1 \cdot L_2$
From L, can deduce L^*

Fact: Every regular language is deducible.
I.e., if $\exists$ a DFA deciding L, then you can deduce L.

Proving this fact is also a little tricky.

Regular Expressions

A **regular expression** over Σ (say, $\{a,b\}$) is something that looks like this:

$$a(a \cup b)^*a \cup b(a \cup b)^*b \cup a \cup b$$

It is a syntactic representation of the deduction of a regular language in the Deductive System.

Also stands for the deduced language; e.g., the regular expression above stands for $\{x \in \{a,b\}^* : x \text{ starts \& ends with same char}\}$.

Regular Expressions

Commonly used in string searching (e.g., *grep*).

You’ll also see some shorthands in practice:

$\vert$	instead of $\cup$
R^+	for RR^*
ε	for $\varnothing^*$
Σ or .	for the union of all single characters
R^n	for $RRR \cdots R$ (n times)
R?	for $(R \vert \varepsilon)$

and more…

String Searching

The *simplest* string searching problem:

Instance: Text T, length n. Substring w, length k.

Solution: Yes/No: Does w occur in T?

Naive algorithm:

$$T = \boxed{a_1, a_2, a_3,} a_4, a_5, \ldots, a_n$$

Running time: about nk steps

String Searching

Instance: Text T, length n. Substring w, length k.

Solution: Yes/No: Does w occur in T?

Automaton solution:

The language $\Sigma^* w \Sigma^*$ is regular!

There is some DFA M_w which decides it.

Once you build M_w, feed in T:

running time is about n steps!

Time to build M_w?

There's a simple alg. running in ~ k^3 steps.

String Searching

Instance: Text T, length n. Substring w, length k.

Solution: Yes/No: Does w occur in T?

Automaton solution:

The language $\Sigma^* w \Sigma^*$ is regular!

There is some DFA M_w which decides it.

Once you build M_w, feed in T:

running time is about n steps!

Time to build M_w?

Knuth–Morris–Pratt '77: # steps ~ k

String Searching

Instance: Text T, length n. Substring w, length k.

Solution: Yes/No: Does w occur in T?

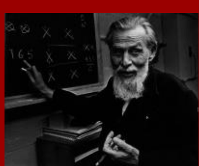


Pittsburgh native,
CMU bachelor's,
CMU professor.

Knuth–Morris–Pratt '77: # steps ~ k

Finite automata — to the max

Finite automata were first studied in the 1940's in the context of neurophysiology.




McCulloch & Pitts

Finite automata — to the max

'40s & '50s: further studied by mathematicians, linguists, electrical engineers

1959: DFAs codified & this lecture's results proved by Michael Rabin & Dana Scott


CMU prof. emeritus

Finite automata — to the max

Rabin & Scott also invented DFAs with certain **“magical superpowers”** which you’ll investigate on the homework.

Actually, they showed that adding these superpowers does not increase the set of languages accepted by DFAs.

For this they won the **Turing Award**.

Finite automata — to the max

A further generalization of DFAs: “nondeterministic pushdown automata”.

These decide the “context-free languages”.

A further further generalization: “linear bounded automata”.

These decide the “context-sensitive languages”.

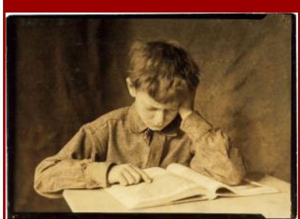
Finite automata — to the max

A further further further generalization: “Turing Machines”.

These decide the “decidable languages”.

We discuss them in the next lecture!

Study Guide



Definitions:
- Problems, instances, strings, languages.
- DFAs.
- Regular operations.
- Regular expressions.

Theorem/proof:
- 0^n1^n is not regular.
- Union Theorem.

Practice:
- Building/analyzing DFAs.