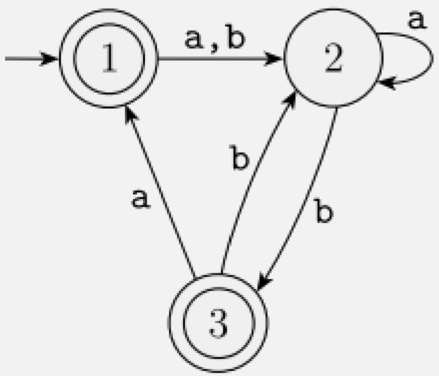
CS622

Regular Languages and Finite Automata

Monday, September 13, 2021

UMass Boston Computer Science



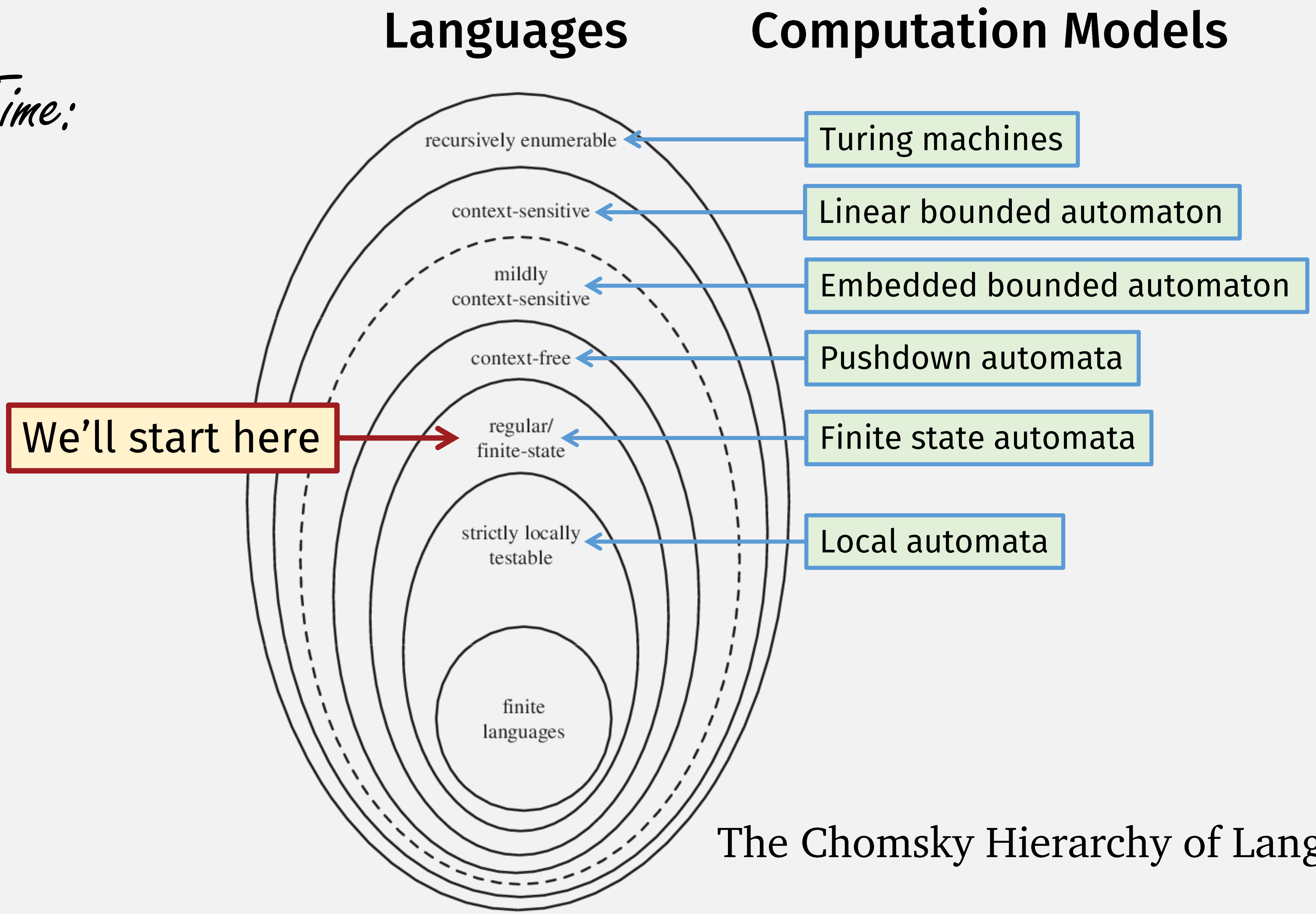
Logistics

- HW1 released
  - Due Sun 9/19 11:59pm on gradescope
  - (Make sure your gradescope account is active and works!)

Last Time: Formal Definition of a Language

- A **language** is a (possibly infinite) set of *strings*
  - E.g., the set of all binary numbers
- A **string**/word is a (finite) sequence of chars from an *alphabet*
  - E.g., `010101`
- An **alphabet** is a (finite, non-empty) set of chars/symbols
  - E.g., {`0`, `1`} (binary digits, the alphabet of computers)

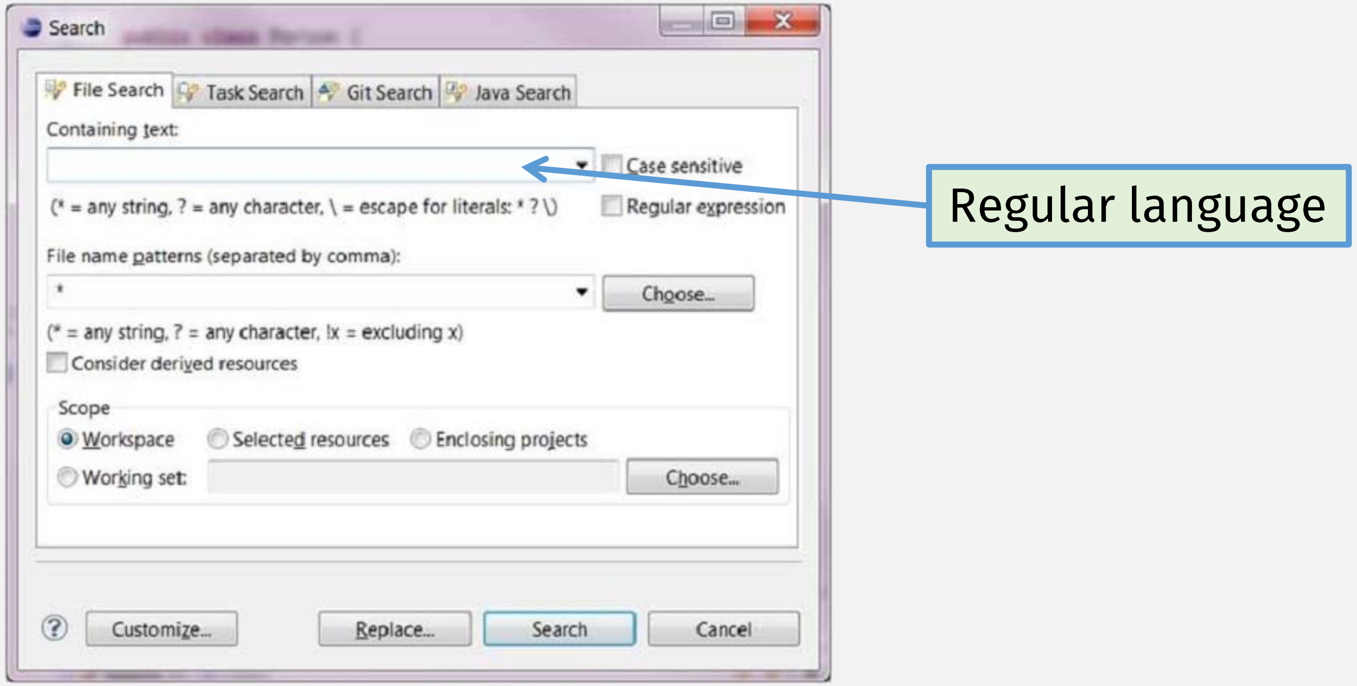
Last Time:



The Chomsky Hierarchy of Languages

"Regular" Languages

- Commonly used in search and text processing tools
  - E.g., `grep`, `sed`, `awk`



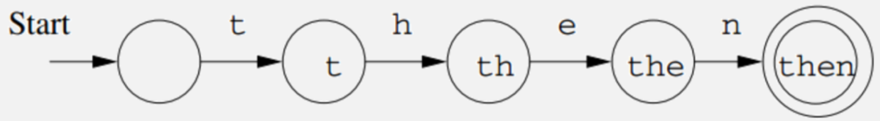
A regular language is recognized by a **finite state automaton** computer

Finite State Automaton

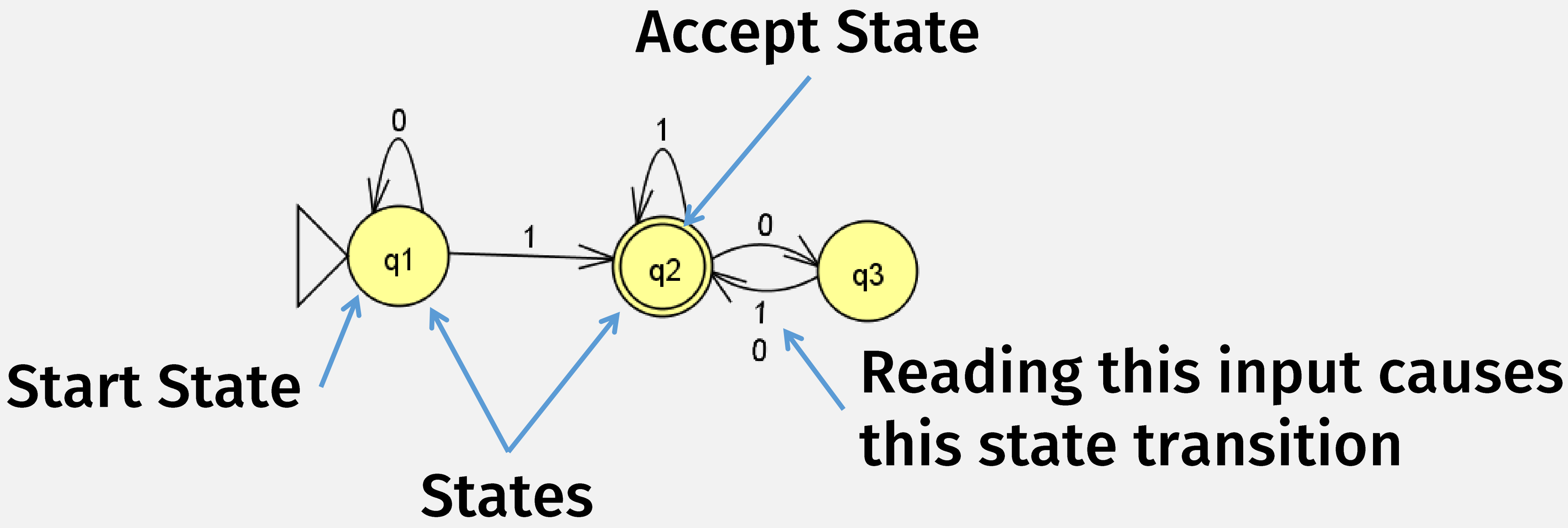
- A.k.a., "finite automaton",
  "finite state machine" (FSM),
  "deterministic finite state automaton" (DFA)

- <u>Key characteristic</u>:
  - Has a **finite** number of states
  - I.e., it's a computer with a finite amount of memory
    - Can't dynamically allocate

- Often used for text matching



Finite Automata: State Diagram



Finite Automata: The Formal Definition

5 components

DEFINITION 1.5

A ***finite automaton*** is a 5-tuple $\boxed{(Q, \Sigma, \delta, q_0, F)}$, where

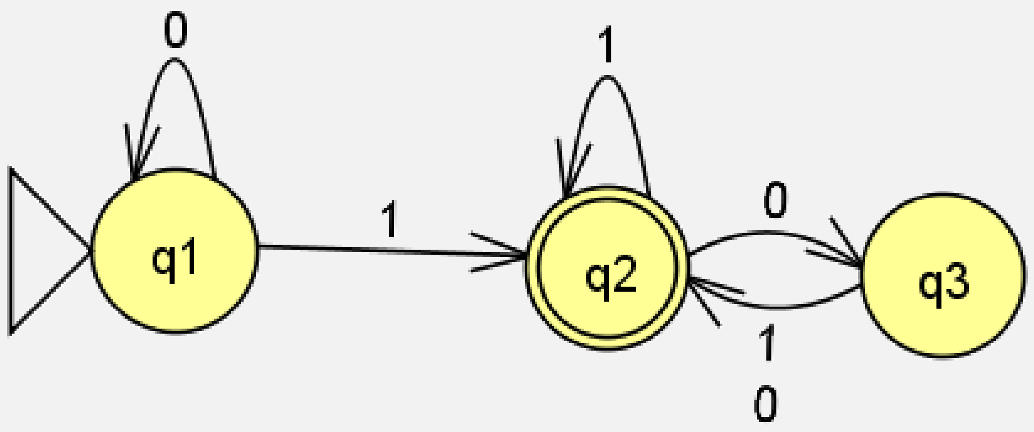
1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta \colon Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

State Diagram vs Formal Description

DEFINITION 1.5

A ***finite 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. $\delta : Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.



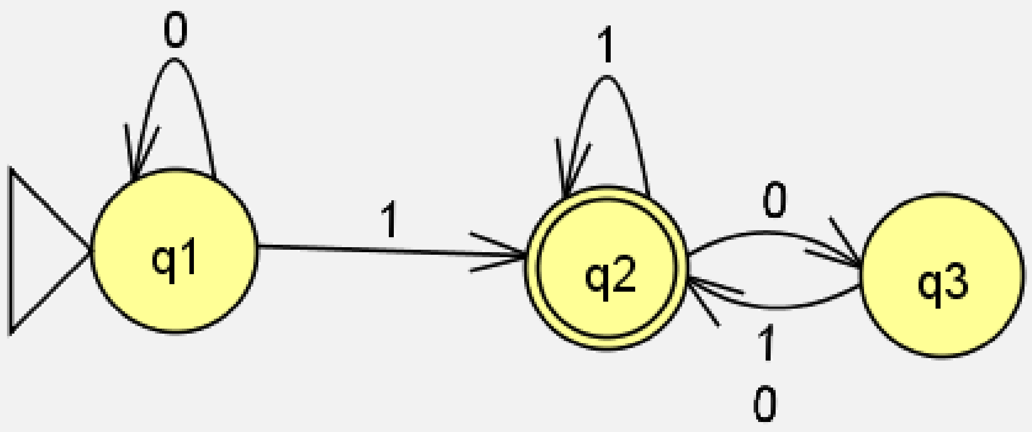
DEFINITION 1.5

A ***finite 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. $\delta\colon Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

Formal description

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



State diagram

DEFINITION 1.5

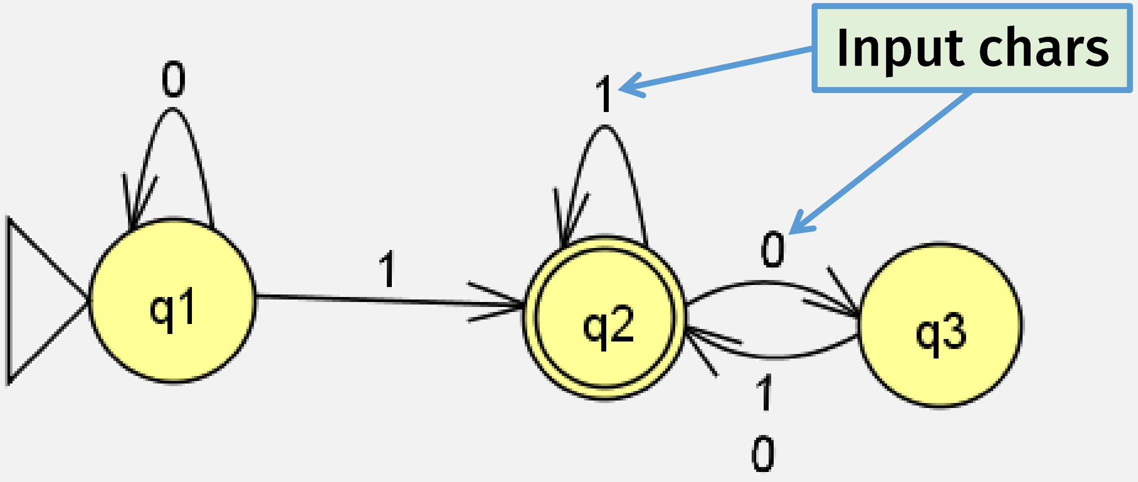
A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

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

1. $Q = \{q_1, q_2, q_3\}$,

rs



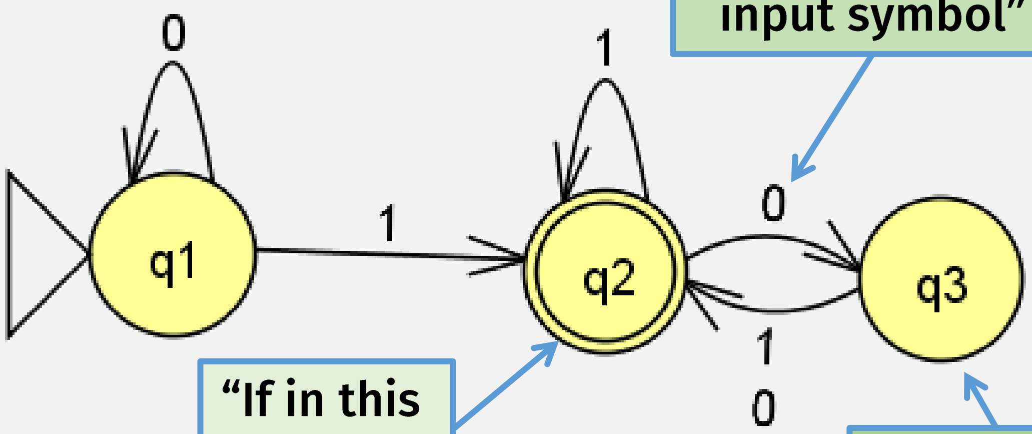
DEFINITION 1.5

A ***finite 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. $\delta : Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

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

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



“And this is next input symbol”

“If in this state”

“Then go to this state”

his is next symbol”

n go to state”

DEFINITION 1.5

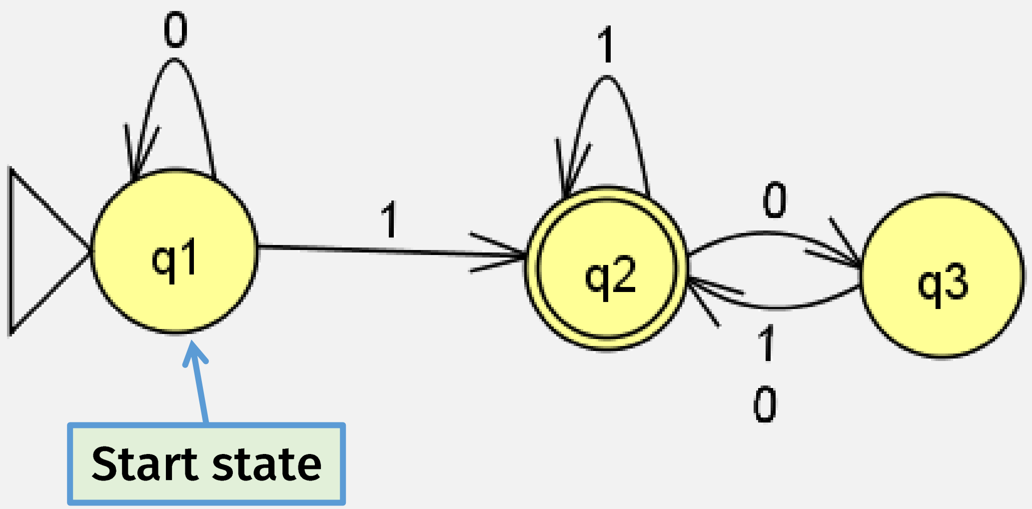
A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta \colon Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

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

1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as

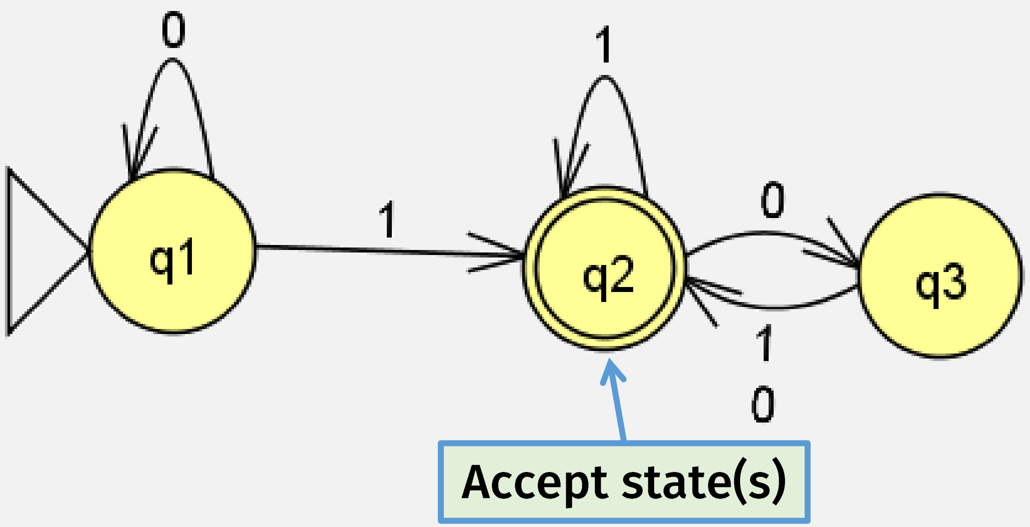
	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,



DEFINITION 1.5

A ***finite 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. $\delta\colon Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.



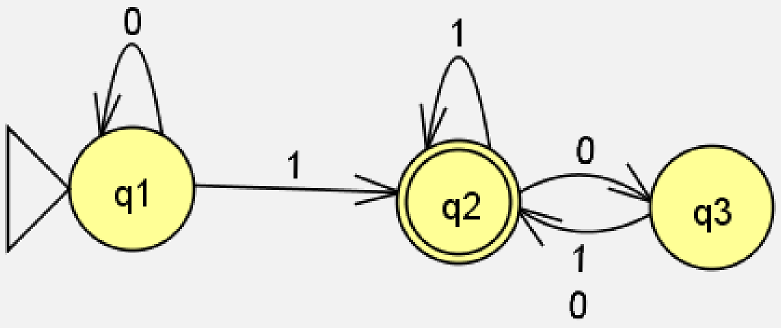
$M_1 = (Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2,

4. q_1 is the start state, and

“Running” an FSM “Program” (JFLAP demo)

- **Computer:** 

- **Program:** “`1101`”

$\delta : Q \times \Sigma \longrightarrow Q$ is the ***transition function***

Running an FSM Program: Formal Model

Define the extended transition function: $\hat{\delta} : Q \times \Sigma^* \to Q$

- *Inputs*:
  - Some beginning state $q \in Q$ (not necessarily the start state)
  - Input string $w = a_1 a_2 \cdots a_n$ where $a_i \in \Sigma$
- *Output*:
  - Some ending state (not necessarily an accept state)

(Defined recursively, on the length of the input string)

- Base case: $\hat{\delta}(q, \epsilon) = q$
- Recursive case: $$\hat{\delta}(q, w'a) = \underbrace{\delta}_{\text{Single transition step}}(\underbrace{\hat{\delta}}_{\text{Recursive call}}(q, \overset{\text{First chars}}{w'}), \overset{\text{Last char}}{a})$$

FSM Computation Model: Summary

Informally	*Formally*
• Computer = a finite automata	• $M = (Q, \Sigma, \delta, q_0, F)$
• Program = input string of chars	• $w = w_1 w_2 \cdots w_n$
To run a program:	
• Start in "start state"	• $r_0 = q_0$
• Read 1 char at a time, changing states according to transition table	• $\delta(r_i, w_{i+1}) = r_{i+1},\ \text{for } i = 0, \ldots, n-1$ • Or $\hat{\delta}(r_0, w)$
• Result = • "Accept" if last state is "Accept" state • "Reject" otherwise	• M ***accepts*** w if $\hat{\delta}(q_0, w)$ is in F

A *language* is a set of strings.

A Finite Automaton's Language

- A machine M ***accepts*** w if $\hat{\delta}(q_0, w)$ is in F

- Language of M = $L(M)$ = $\{w \mid M \text{ accepts } w\}$

"the set of all ..."

"such that ..."

A language is called a ***regular language*** if some finite automaton recognizes it.

A ***language*** is a set of strings.

M ***recognizes language*** A if $A = \{w|\ M \text{ accepts } w\}$

Is it Regular?

- <u>If given</u>: Finite Automata M
  - <u>We know</u>: the language recognized by M is a regular language
- <u>If given</u>: some Language A
  - Is A is a regular language?
    - Not necessarily
  - How do we determine, i.e., *prove*, that A is a regular language?

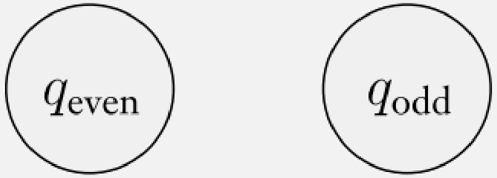
A language is called a ***regular language*** if some finite automaton recognizes it.

Designing Finite Automata: Tips

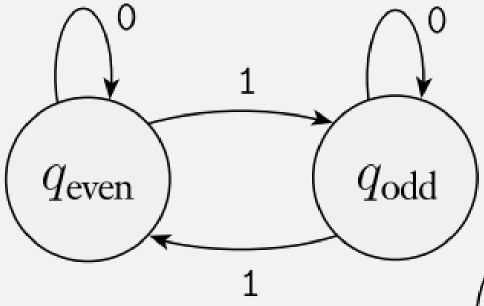
- Input may only be read once, one char at a time
- Must decide accept/reject after that
- States = the machine's **memory**!
  - Machine has finite amount of memory, and must be allocated in advance
  - So think about what information must be remembered.
- Every state/symbol pair must have a transition (for DFAs)
- <u>Example</u>: a machine that accepts strings with odd number of 1s

Design a DFA: accepts strs with odd # **1**s

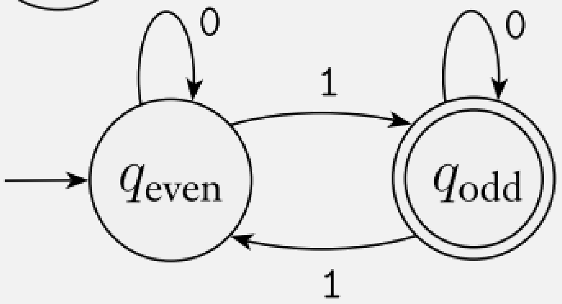
- States:
  - 2 states:
    - seen even **1**s so far
    - seen odds **1**s so far



- Alphabet: `0` and `1`
- Transitions:



- Start / Accept states:



Is our machine "correct"?

We have to prove it!

Proof by Induction

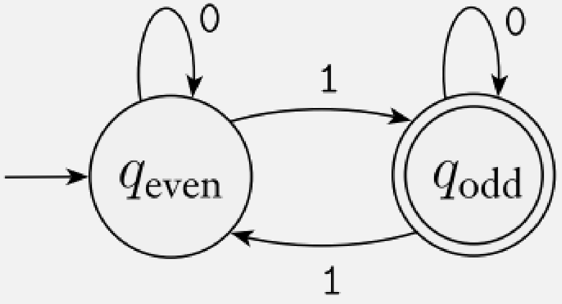
- To prove that a **property** P is true for a **thing** x:
  1. Prove P for the base case of x (usually easy)
  2. Prove P for the inductive case :
     - Assume the induction hypothesis (IH):
       - Assume $\mathrm{P}(\mathrm{x}_{\mathrm{smaller}})$ true for some measure of “smaller”
       - E.g., if x is string, then “smaller” = length of string
     - Use IH to prove $\mathrm{P}(\mathrm{x})$
       - Usually involves a case analysis on all possible ways to go from $\mathrm{x}_{\mathrm{smaller}}$ to x

- Why can we assume IH is true???
  - Because we can always start at base case,
  - Then use it to prove for slightly larger case,
  - Then use that to prove for slightly larger case …

Odd # **1**s DFA: Proof of Correctness

$\mathrm{P}(\mathrm{x})$ = M **accepts strings** x **with odd # of 1s, else rejects**

- <u>Base case</u> (the smallest string):
  - Let $\mathrm{x} = \varepsilon$ (the empty string!)
  - x **has even 1s** and M **rejects, so** $\mathrm{P}(\mathrm{x}) = \textsc{True}$

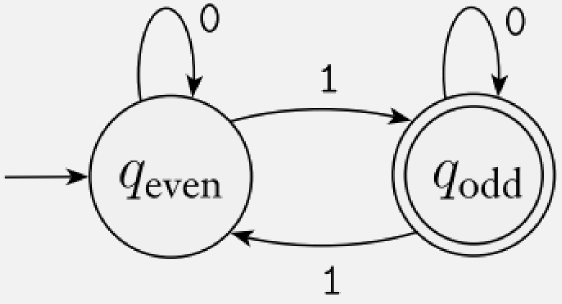


DFA M

Odd # 1s DFA: Proof of Correctness

P(x) = *M* accepts strings x with odd # of 1s, else rejects

- Induction step:
  - Let x = x'a, where a = `0` or 1
  - Induction Hypothesis (IH): Assume P(x') = TRUE
  - Use P(x') to prove P(x), analyzing all possible ways to get x from x':
    - If x' has odd # 1s, then *M* is in state q_{odd}:
      - Let x = x'`0`: *M* stays in q_{odd} and accepts, so P(x) = TRUE
      - Let x = x'1: *M* goes to q_{even} and rejects, so P(x) = TRUE
    - If x' has even # 1s, then *M* is in state q_{even}:
      - Let x = x'`0`: *M* stays in q_{even} and rejects, so P(x) = TRUE
      - Let x = x'1: *M* goes to q_{odd} and accepts, so P(x) = TRUE ■



DFA *M*

Thus we have proven that machine *M* recognizes the language of strings containing an odd # 1s

"Q.E.D."

Next time: Combining DFAs

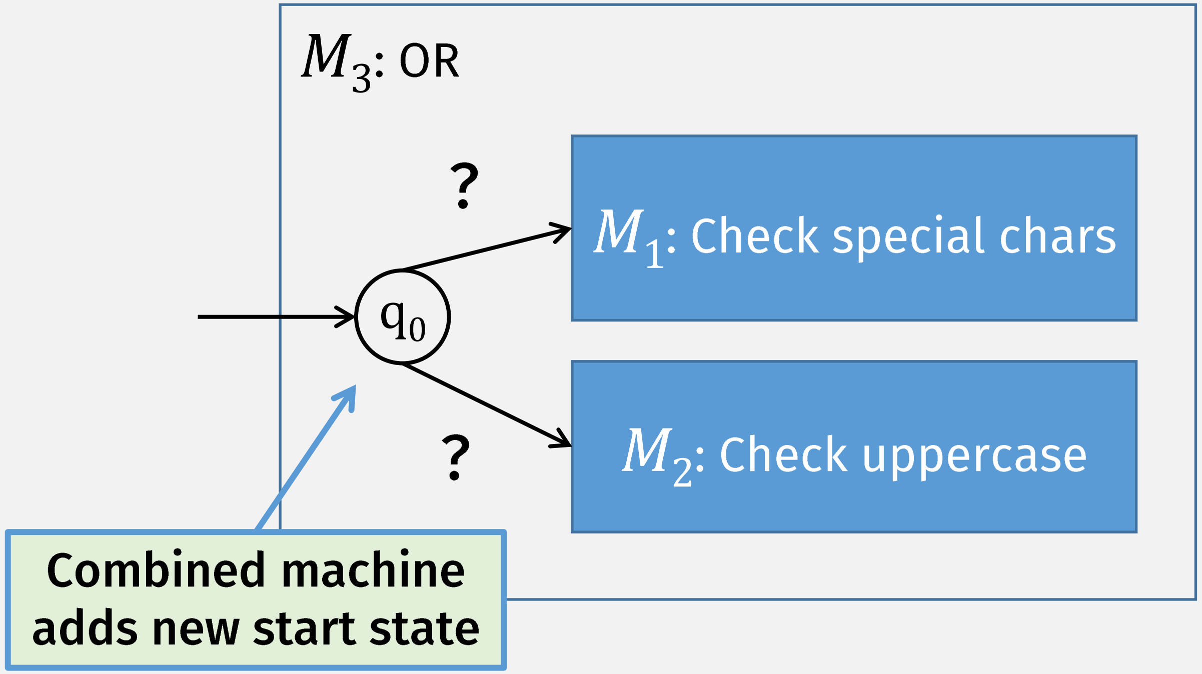
From: https://www.umb.edu/it/password

Password Requirements

- Passwords must have a minimum length of ten (10) characters - but more is better! ← DFA
- Passwords **must include at least 3** different types of characters:
  - upper-case letters (A-Z) ← DFA
  - lower-case letters (a-z) ← DFA
  - symbols or special characters (%, &, *, $, etc.) ← DFA
  - numbers (0-9) ← DFA
- Passwords cannot contain all or part of your email address ← DFA
- Passwords cannot be re-used ← DFA

It would be nice if we could just combine them all together into one big DFA!

Next time: Combining DFAs



Problem:
Once we enter one of the machines, we can't go back to the other one!

Solution:
Nondeterminism: allows being in multiple states, i.e., multiple machines, at once!