CS420

Computing With Finite Automata

Tuesday, September 13, 2022

UMass Boston Computer Science

Announcements

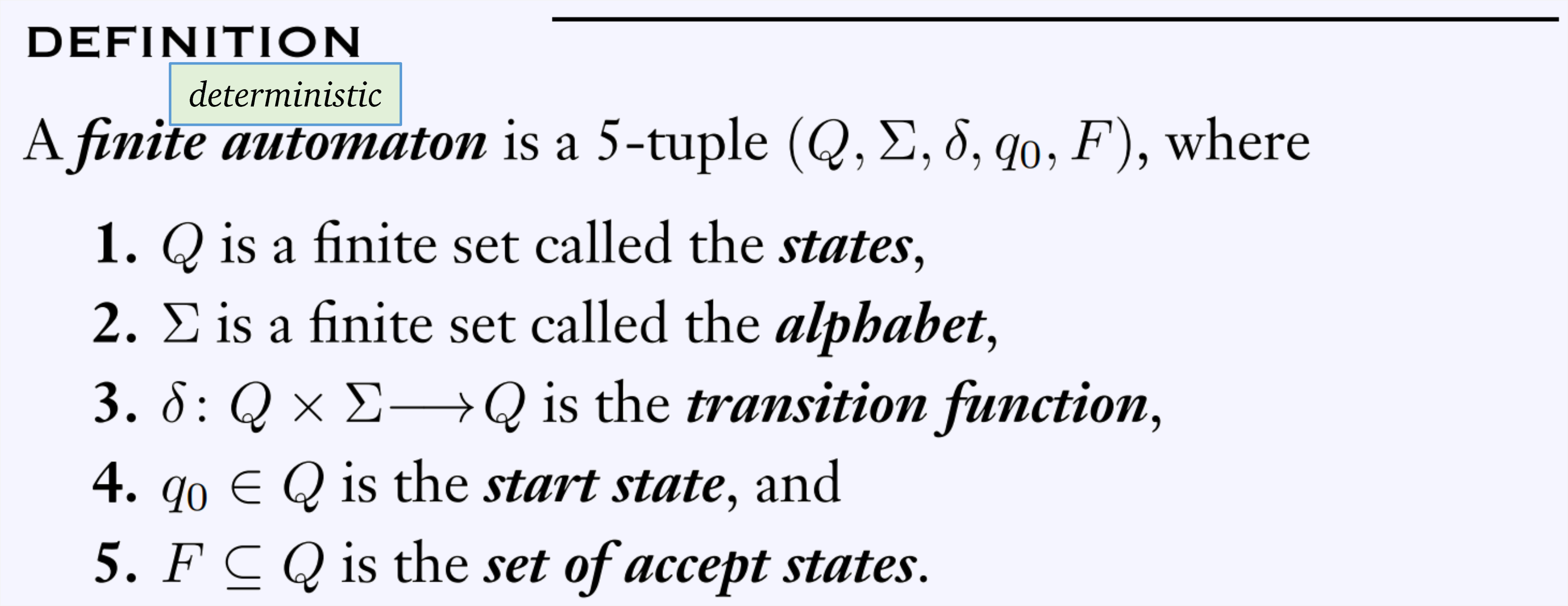
- HW 0 in extended, due Wed 11:59pm

- HW 1 released soon

- Please ask all HW questions on Piazza!
  - So all course staff can see,
  - and entire class can benefit
  - Please do not directly email course staff with HW questions

- TA: Sean Rasku-Casas
  - Office Hours Mondays 12:30-2pm, in the TA room (McCormack 3rd floor)

Last Time: Finite Automata Formal Definition



DEFINITION

deterministic

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***.

Also called a **Deterministic Finite Automata (DFA)**

(will be important later)

Last Time: FSM Computation Rules

HINT: to better understand the math, always work out individual examples

Informally

- "Program" = a finite automata
- Input = string of chars, e.g. "`1101`"

To run a "program":

- Start in "start state"
- Repeat:
  - Read 1 char;
  - Change state according to the transition table
- Result =
  - **Accept** if last state is "Accept" state
  - **Reject** otherwise

Formally (i.e., mathematically)

- $M = (Q, \Sigma, \delta, q_0, F)$
- $w = w_1 w_2 \cdots w_n$

Define variables r_i, $i = 0 \dots n$, representing sequence of states in the computation

- $r_0 = q_0$

e.g., i=1, $r_1 = \delta(r_0, w_1)$

$r_2 = \delta(r_1, w_2)$ …

- $r_i = \delta(r_{i-1}, w_i)$, for $i = 1, \dots, n$

Let's come up with **nicer notation** to represent this part

- M ***accepts*** w if

  sequence of states $r_0, r_1, \dots, r_n$ in Q exists …

  with $r_n \in F$

This is still a little verbose / informal

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

An Extended Transition Function

$* =$ "0 or more"

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

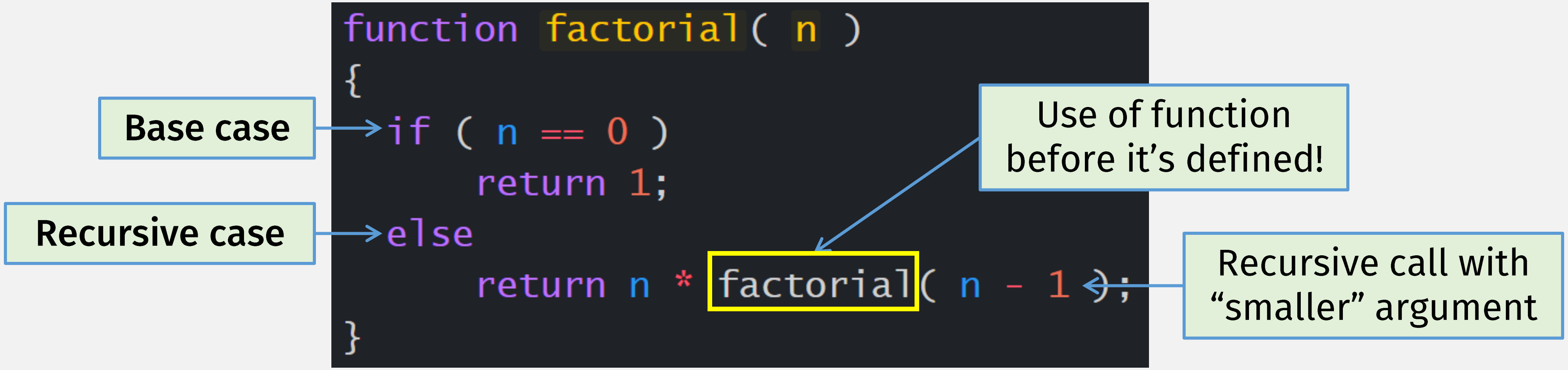
Σ^* = set of all strings!

- Domain:
  - **Beginning state** $q \in Q$ (not necessarily the start state)
  - **Input string** $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$
- Range:
  - **Ending state** (not necessarily an accept state)

(Defined recursively)

- Base case: …

Recursive Definitions



```
function factorial( n )
{
  if ( n == 0 )
        return 1;
  else
        return n * factorial( n - 1 );
}
```

Base case

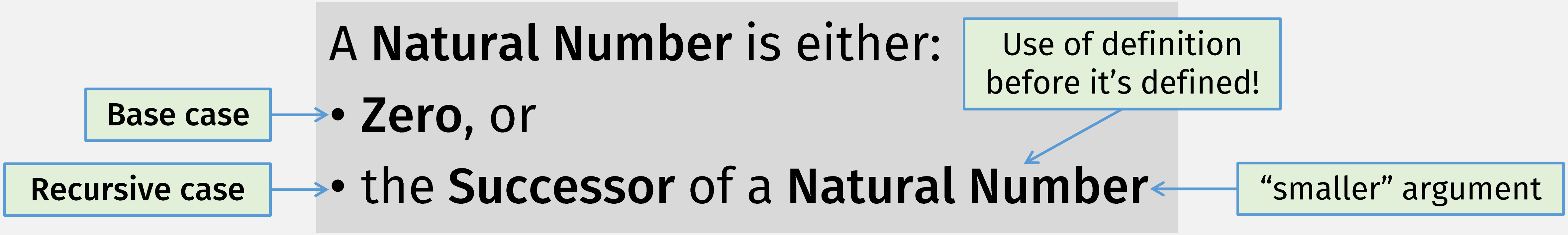
Recursive case

Use of function before it's defined!

Recursive call with "smaller" argument

- Why is this allowed?
  - It's a **feature** (i.e., an **axiom**) of the **programming language**
- Why does this work?
  - Because the **recursive call always has a "smaller" argument …**
  - … and so **eventually reaches the base case and stops**

Recursive Definitions



A **Natural Number** is either:

- **Zero**, or
- the **Successor** of a **Natural Number**

Examples

- **Zero**
- **Successor** of **Zero** (= "one")
- **Successor** of **Successor** of **Zero** (= "two")
- **Successor** of **Successor** of **Successor** of **Zero** (= "three") …

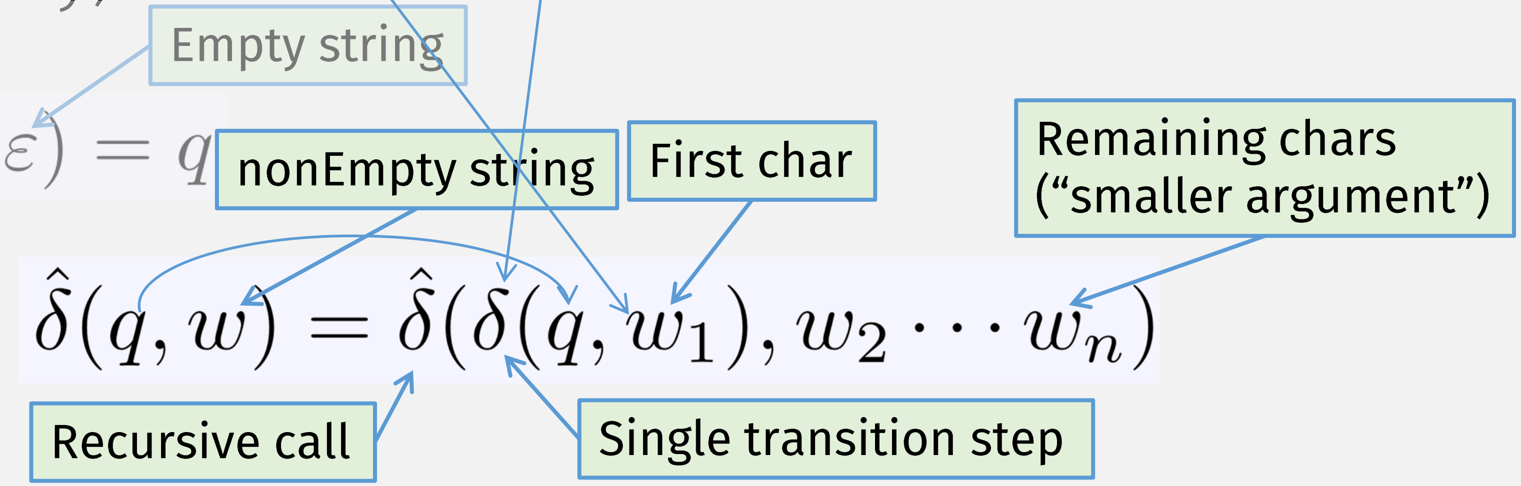
$\delta : Q \times \Sigma \longrightarrow Q$ is the ***transition function***

An Extended Transition Function

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

- Domain:
  - **Beginning state** $q \in Q$ (not necessarily the start state)
  - **Input string** $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$
- Range:
  - **Ending state** (not necessarily an accept state)

(Defined recursively)



Empty string

- Base case: $\hat{\delta}(q, \varepsilon) = q$

nonEmpty string

First char

Remaining chars
(“smaller argument”)

- **Recursive case:** $\hat{\delta}(q, w) = \hat{\delta}(\delta(q, w_1), w_2 \cdots w_n)$

Recursive call

Single transition step

FSM Computation Model

Informally

- "Program" = a finite automata
- Input = string of chars, e.g. "`1101`"

To run a "program":

- Start in "start state"
- Repeat:
  - Read 1 char;
  - Change state according to the transition table
- Result =
  - "**Accept**" if last state is "Accept" state
  - "**Reject**" otherwise

Formally (i.e., mathematically)

- $M = (Q, \Sigma, \delta, q_0, F)$
- $w = w_1 w_2 \cdots w_n$
- $r_0 = q_0$
- $r_i = \delta(r_{i-1}, w_i)$, for $i = 1, \ldots, n$

Let's come up with **nicer notation** to represent this part

- M ***accepts*** w if

  sequence of states $r_0, r_1, \ldots, r_n$ in Q exists . . .

  with $r_n \in F$

FSM Computation Model

Informally

- “<u>Program</u>” = a finite automata
- <u>Input</u> = string of chars, e.g. “`1101`”

To run a “program”:

- <u>Start</u> in “start state”
- <u>Repeat</u>:
  - <u>Read</u> 1 char;
  - <u>Change</u> state according to the <u>transition</u> table
- <u>Result</u> =
  - “**Accept**” if last state is “Accept” state
  - “**Reject**” otherwise

Formally (i.e., mathematically)

- $M = (Q, \Sigma, \delta, q_0, F)$
- $w = w_1 w_2 \cdots w_n$
- $r_0 = q_0$
- $r_i = \delta(r_{i-1}, w_i)$, for $i = 1, \ldots, n$
- M ***accepts*** w if $\hat{\delta}(q_0, w) \in F$

sequence of states $r_0, r_1, \ldots, r_n$ in Q exists . . .

with $r_n \in F$

Definition of Accepting Computations

An **accepting computation,** for FSM $M = (Q, \Sigma, \delta, q_0, F)$ and **string** w:

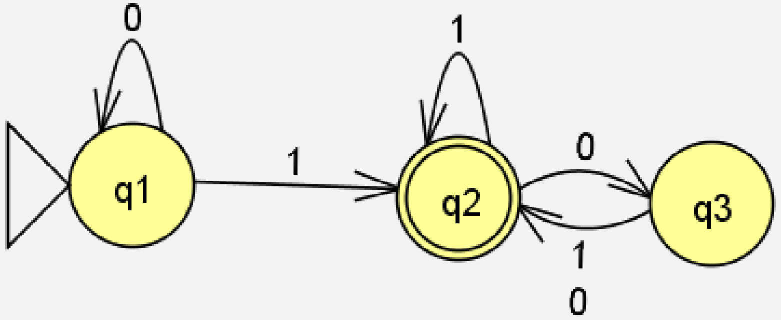
1. starts in the start state q_0
2. goes through a valid sequence of states according to δ
   - this implies that all $w_i \in \Sigma$
3. ends in an accept state

All 3 must be true for a computation to be an **accepting computation!**

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

Accepting Computation or Not?

FSM:



- $\hat{\delta}$(q1, **`1101`**)
  - yes
- $\hat{\delta}$(q1, **`110`**)
  - No (doesn’t end in accept state)
- $\hat{\delta}$(q2, **`101`**)
  - No (doesn’t start in start state)
- $\hat{\delta}$(q1, **`123`**)
  - No (doesn’t follow delta transition function)

Languages and Strings

- A **language** is a <u>set</u> of **strings**
- A **string** is a <u>finite</u> <u>sequence</u> of **symbols** from an **alphabet**
- An **alphabet** is a <u>non-empty finite set</u> of **symbols**

$$\Sigma_1 = \{\mathtt{0},\mathtt{1}\}$$

$$\Sigma_2 = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}, \mathtt{h}, \mathtt{i}, \mathtt{j}, \mathtt{k}, \mathtt{l}, \mathtt{m}, \mathtt{n}, \mathtt{o}, \mathtt{p}, \mathtt{q}, \mathtt{r}, \mathtt{s}, \mathtt{t}, \mathtt{u}, \mathtt{v}, \mathtt{w}, \mathtt{x}, \mathtt{y}, \mathtt{z}\}$$

Computation and Languages

- The **language** of a **machine** is the **set of all strings that it accepts**
- E.g., An **FSM** M ***accepts*** w if $\hat{\delta}(q_0, w) \in F$
- **Language** of $M = L(M) = \{w \mid M \text{ accepts } w\}$

“the set of all …”

“such that …”

Language Terminology

- M ***accepts*** w ← string

- M ***recognizes language*** A ← Set of strings

  $$\text{if } A = \{w \mid M \text{ accepts } w\}$$

Computation and Classes of Languages

- The **language** of a machine is the **set of all strings that it accepts**

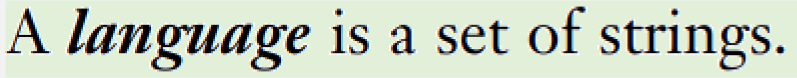
- A **computation model** is equivalent to the set of machines it defines

- E.g., all possible FSMs are a computation model

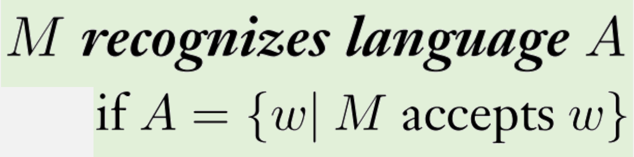
- Thus: a **computation model** is also equivalent to a set of languages

Regular Languages: Definition

If a finite automaton (FSM) recognizes a language, then that language is called a **regular language**.



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



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

A Language, Regular or Not?

- If given: a Finite Automaton M
    - We know: $L(M)$, the language recognized by M, is a regular language

If a finite automaton (FSM) recognizes a language, then that language is called a **regular language**.

- If given: a Language A
    - Is A is a regular language?
        - Not necessarily!
    - How do we determine, i.e., *prove*, that A is a regular language?

Kinds of Mathematical Proof

- Deductive Proof
    - Start with known facts (i.e., **premises**)
    - Use logical **inference rules** to reach new **conclusions**

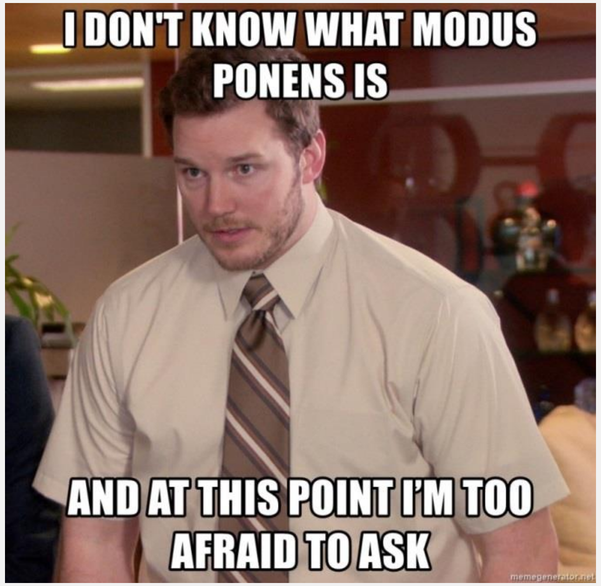
An Inference Rule: Modus Ponens

Premises

- If P then Q
- P is true

Conclusion

- Q must also be true



An Inference Rule: Modus Ponens

Premises

- If P then Q
- P is true

Conclusion

- Q must also be true

Example **Premises**

- If **there is an FSM recognizing language** A, then A **is a regular language**
- **We know an FSM** M **where** $L(M) = A$

Conclusion

- A **is a regular language!**

A Language, Regular or Not?

- If given: **a Finite Automaton** M
  - We know: $L(M)$, the language recognized by M, is a regular language

If **a finite automaton** (FSM) <u>recognizes</u> **a language,** then **that language is called a regular language.**

- If given: **a Language** A
  - **Is** A **is a regular language?**
    - **Not necessarily!**
  - **How do we determine,** i.e., ***prove*****, that** A **is a regular language?**

Create an FSM recognizing A!

Designing Finite Automata: Tips

- Input may only be read once, one char at a time

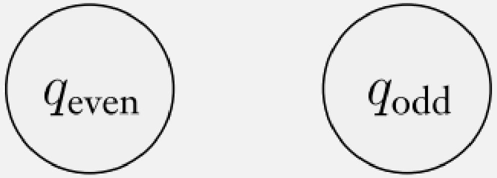
- Must decide accept/reject after that

- States = the machine's **memory**!
  - # states must be decided in advance
  - So think about what information must be remembered.

- Every state/symbol pair must have a transition (for DFAs)

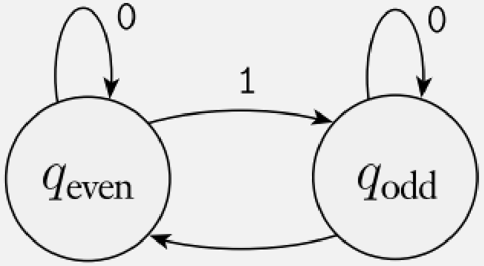
Design a DFA: accept strs with odd # **1**s

- <u>States</u>:
  - 2 states:
    - seen even **1**s so far
    - seen odds **1**s so far

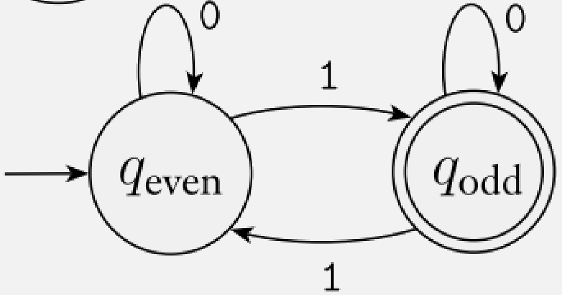


- <u>Alphabet</u>: `0` and `1`

- <u>Transitions</u>:



- <u>Start</u> / <u>Accept</u> states:

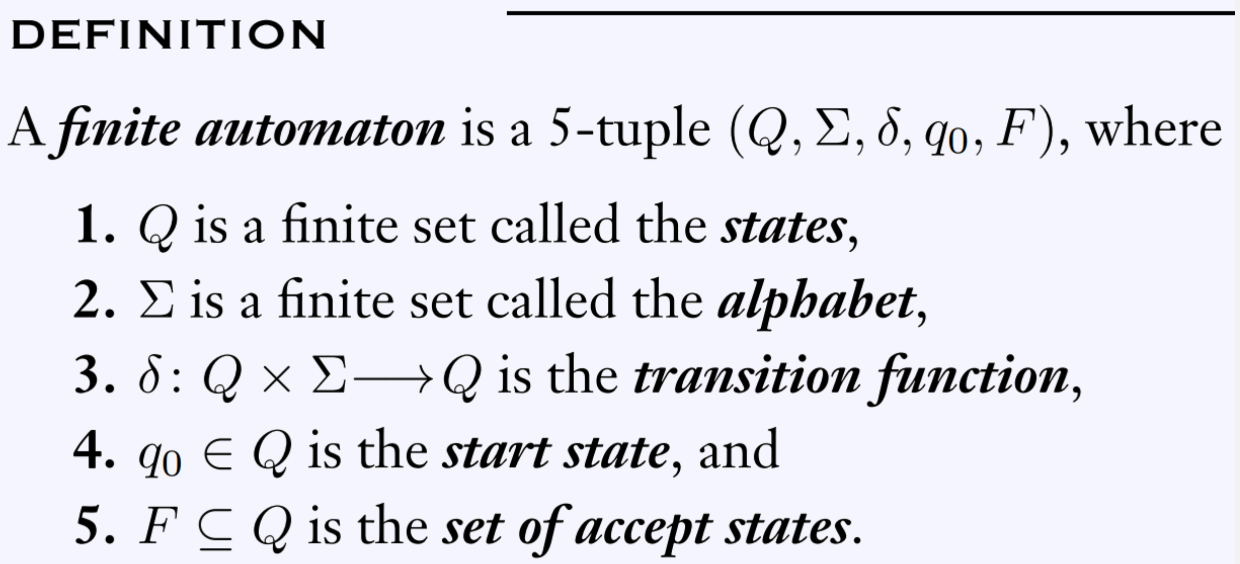


In-class exercise

- <u>Prove:</u> the following language is a regular language:
  - $A = \{w \mid w \text{ has exactly three 1's}\}$
  - i.e., design a finite automata that recognizes it!

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

- Remember:



DEFINITION

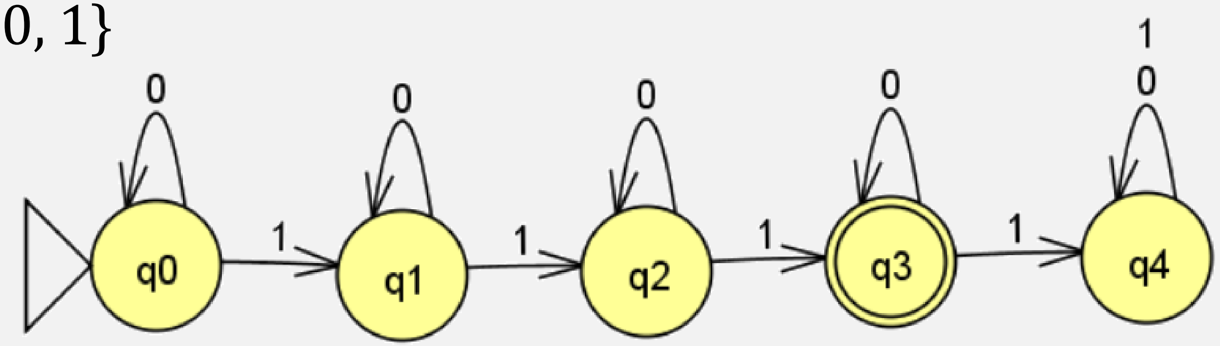
A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

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***.

In-class exercise Solution

- Design finite automata recognizing:
  - $\{w \mid w \text{ has exactly three 1's}\}$

- *States*:
  - Need one state to represent how many 1's seen so far
  - $Q = \{q_0, q_1, q_2, q_3, q_{4+}\}$
- *Alphabet*: $\Sigma = \{0, 1\}$
- *Transitions*:



- *Start state*:
  - q_0
- *Accept states*:
  - $\{q_3\}$

So finite automata are used to <u>recognize simple string patterns</u>?

Yes!

Have you ever used a programming language feature to <u>recognize simple string patterns</u>?

Check-in Quiz 9/13

On gradescope