CS420

Computing With Finite Automata

Monday, January 30, 2023

UMass Boston Computer Science

Announcements

- HW 0
  - due Tue 1/31 11:59pm EST
- HW 1
  - released Wed
- Please ask all HW questions on Piazza!
  - So all course staff can see,
  - and entire class can benefit
  - Do not email course staff with HW questions

Last Time: Finite Automata Formal Definition

DEFINITION

deterministic

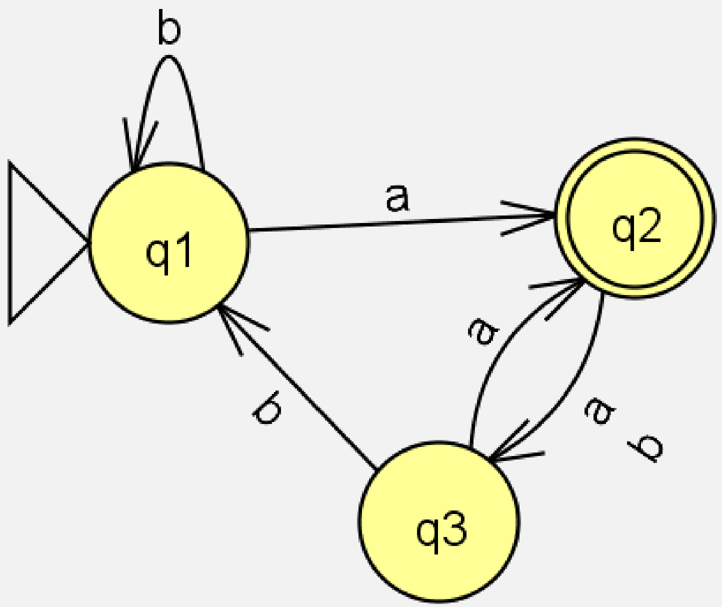
A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

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

Also called a **Deterministic Finite Automata (DFA)**
(will be important later)

In-class Exercise

Come up with a formal description of the following machine:



DEFINITION

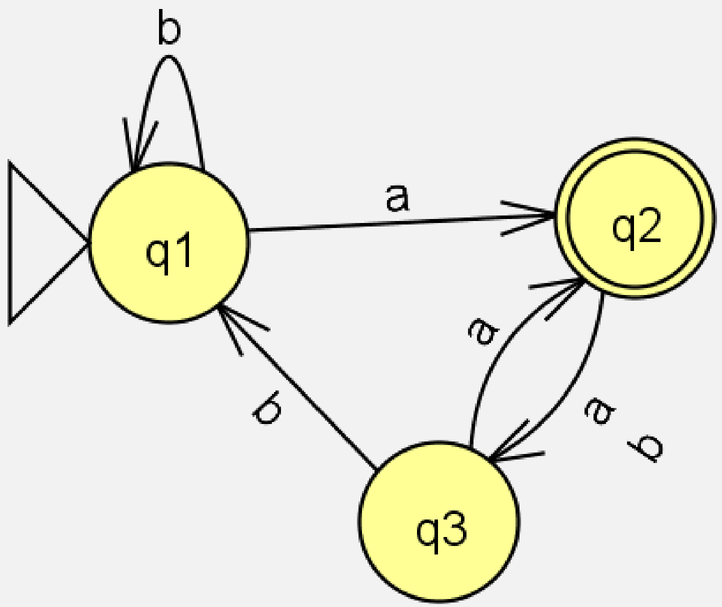
A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

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

- Q = {q1, q2, q3}
- Σ= { **a**, **b** }
- δ
  - δ(q1, **a**) = q2
  - δ(q1, **b**) = q1
  - δ(q2, **a**) = q3
  - δ(q2, **b**) = q3
  - δ(q3, **a**) = q2
  - δ(q3, **b**) = q1
- q_0 = q1
- F = {q2}

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



Quiz Preview

1. Which are possible inputs to an FSM computation?
2. Which are possible results of running an FSM computation?
3. In CS 420, what kind of mathematical object is a language?

A Computation Model is … (from lecture 1)

- Some **base definitions and axioms …**

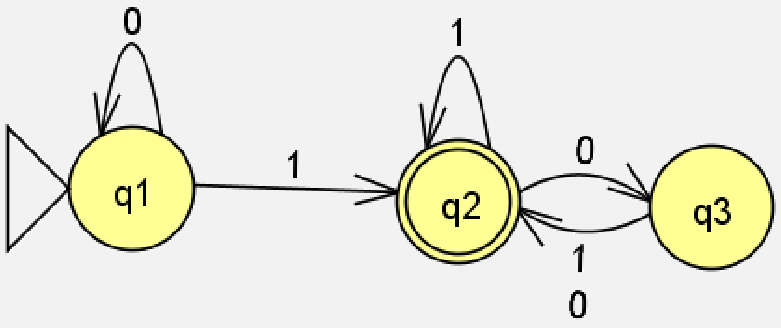
DEFINITION

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

- And **rules that use the definitions …**

Computation with FSMs (JFLAP demo)

- **FSM:** 

- **Input:** “`1101`”

FSM Computation Rules

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

Informally

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

To run a computation / "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 $r_0, \ldots, r_n$, 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 $i = 1, \ldots, n$

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 $r_n \in F$

This is still a little verbose / informal

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

An Extended Transition Function

set of pairs

* = “0 or more”

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

- 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)

Σ^* = set of all strings!

(Defined recursively)

- Base case: ...

Recursive Definitions

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

Base case

Recursive case

Function is called before it is fully defined!

Recursive call with "smaller" argument

- Why is this <u>allowed</u>?
  - It's a "feature" (i.e., an **axiom!**) of the **programming language**
- Why does this "<u>work</u>"? (Why doesn't it loop forever?)
  - 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**

Base case

Recursive case

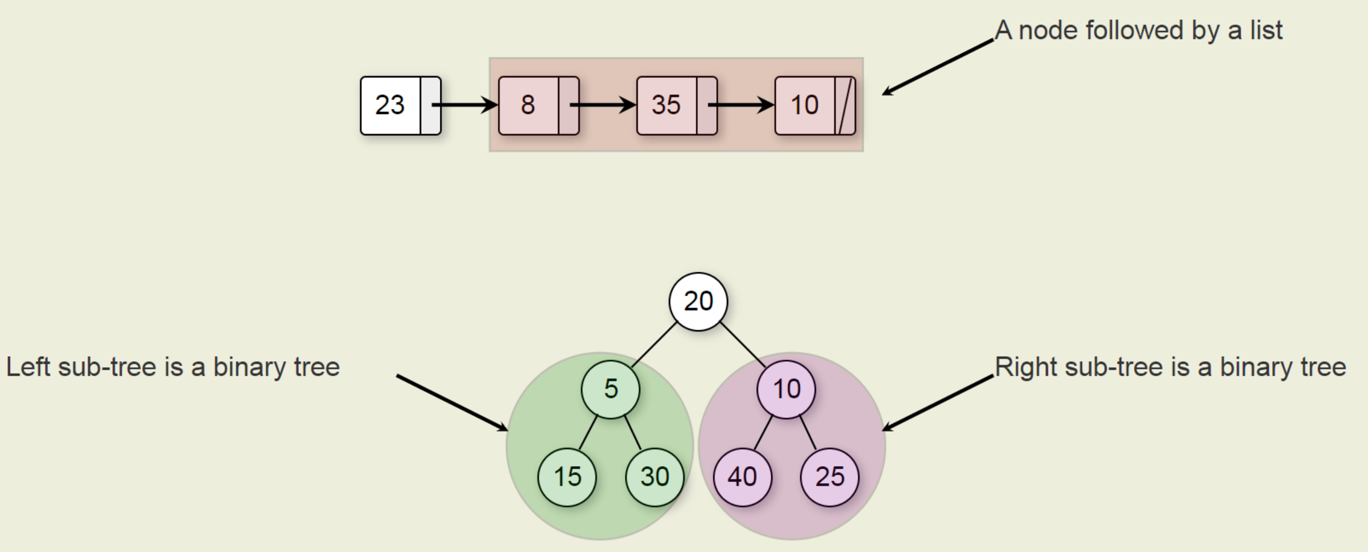
Use of definition before it is fully defined!

"smaller" argument

<u>Examples</u>

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

Recursive Definitions



Recursive definitions have:
- <u>base case</u> and
- <u>recursive case</u>
  (with a "smaller" object)

```
/* Linked list Node*/
class Node {
    int data;
    Node next;
}
```

This is a <u>recursive definition:</u>
Node is used before it is fully defined (but must be "smaller")

$\delta: Q \times \Sigma \longrightarrow Q$ 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	*Formally* *(i.e., mathematically)*

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

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

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

- $r_0 = q_0$
- $r_i = \delta(r_{i-1}, w_i)$, 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:

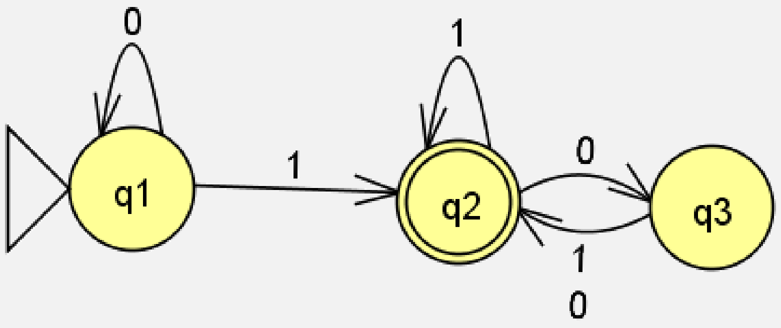
1. starts in the start state q_0
2. goes through a valid sequence of states according to δ
   - this implies that all $w_i \in \Sigma$
3. ends in an accept state

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)

Alphabets, Strings, Languages

- An **alphabet** is a non-empty finite set of symbols

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

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

- A **string** is a finite sequence of symbols from an alphabet

01001 abracadabra ε ← Empty string (length 0)

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

Languages can be infinite

$$A = \{\text{good}, \text{bad}\}$$

$$\emptyset \quad \{\ \}$$

Empty set is a language

$$A = \{w \mid w \text{ contains at least one 1 and an even number of 0s follow the last 1}\}$$

"the set of all ..."

"such that ..."

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

  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) <u>recognizes</u> a language,
> then that language is called a **regular language**.

(modus ponens)

- 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?

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**
- **There is 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

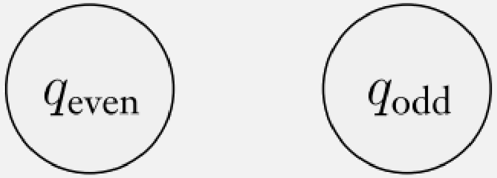
- Input is read only once, one char at a time
- Must decide accept/reject after that
- States = the machine's **memory**!
  - # states must be decided in advance
  - Think about what information must be remembered.
- Every state/symbol pair must have a transition (for DFAs)
- Come up with examples!

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

- On input `1`:
  - **Accept**
- On input `0`:
  - **Reject**
- On input `01`:
  - **Accept**
- On input `11`:
  - **Reject**
- On input `1101`:
  - **Accept**
- On input ε
  - **Reject**

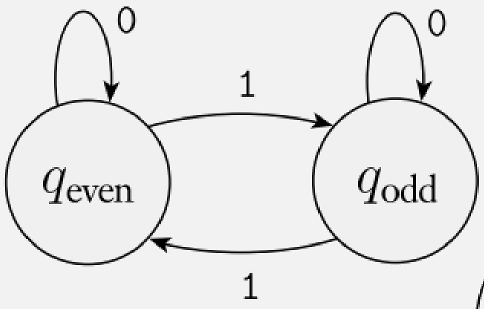
Design a DFA: accept strs with odd # **1**s

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

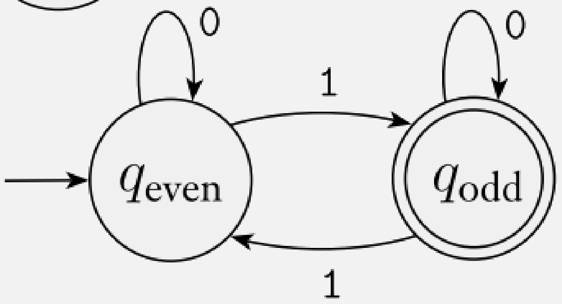


- <u>Alphabet</u>: **0** and **1**

- <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 | w has exactly three 1's}
  - i.e., design a finite automata that recognizes it!

- Where Σ= {0, 1},

- Remember:

DEFINITION

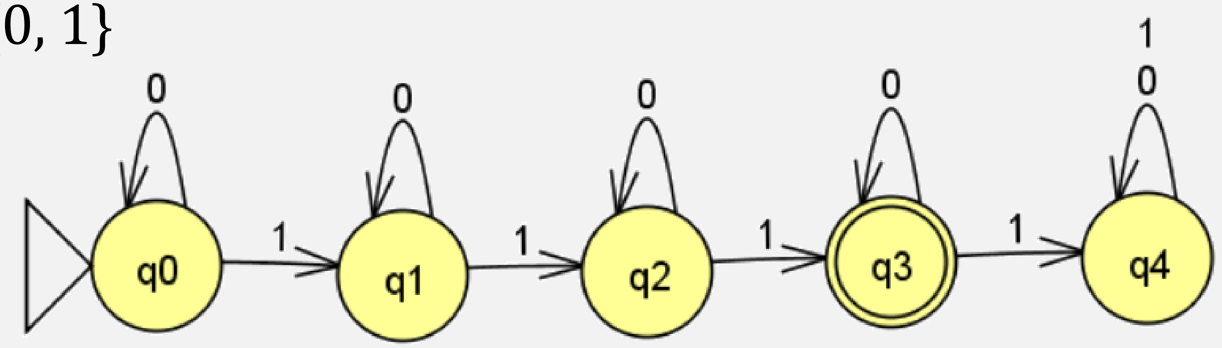
A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

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:
  - $\{w \mid w \text{ has exactly three } 1\text{'s}\}$

- *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 1/30

On gradescope