Designing Finite Automata

Follow us on social media!

@wsucollegeloops

Next Meeting on Feb. 3rd
Career Fair Prep Workshop!

girls who code COLLEGE LOOPS

Join the Discord Server!

SCAN ME

Meetings on Thursdays from 5:30pm-6:30pm!
@CUE 319

Recall definition of Finite Automaton

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***,[1]
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.[2]

footnote

1: note the use of the Cartesian product

2: accept states are also called **final states**

Recall formal definition of computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \cdots w_n$ be a string where each w_i is a member of the alphabet Σ. Then M ***accepts*** w if a sequence of states $r_0, r_1, \ldots, r_n$ in Q exists with three conditions:

1. $r_0 = q_0$,
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \ldots, n-1$, and
3. $r_n \in F$.

Condition 1 says that the machine starts in the start state. Condition 2 says that the machine goes from state to state according to the transition function. Condition 3 says that the machine accepts its input if it ends up in an accept state. We say that M ***recognizes language*** A if $A = \{w \mid M \text{ accepts } w\}$.

Regular languages

DEFINITION 1.16

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

Designing Finite Automata

- Design is a creative process

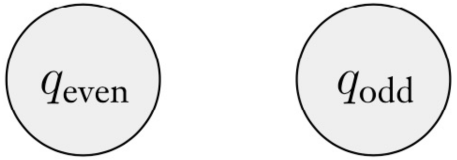
- Helpful approach for designing automata
  - Pretend you are the automata. How would you go about carrying out the task of recognizing a language?
  - You get to see the input string a symbol at a time
  - You must decide whether the string seen so far is in the language
  - Must be ready with an answer since you don't know when the end of the string is coming
  - Need to figure out *what you need to remember*
    - *Have finite memory memory available*

Example 1

- Suppose the alphabet is $\{0,1\}$. *Construct a finite automaton E_1 that recognizes the language consisting of all strings with odd number 1s.*
- Do you need to remember the entire string seen so far in order to determine whether the number of 1s is odd?
- Of course not. It suffices to remember whether the number of 1s seen so far is *even* or *odd* and to keep track of that information as new input is read.

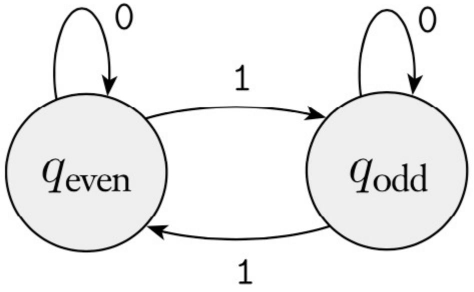
Example 1

Once you determine the necessary information to remember, represent the information as a finite list of possibilities:

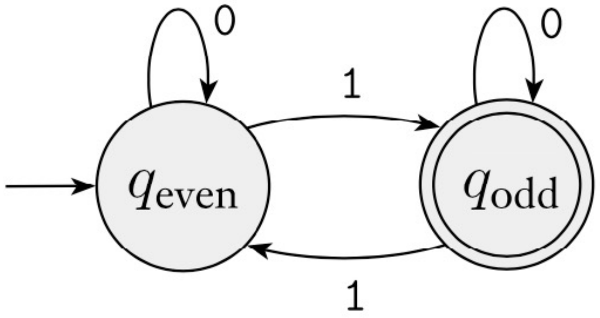


1. **even so far**, and
2. **odd so far**

Next, assign the **transitions** by seeing how to go from one possibility to another upon reading a symbol



Finally, determine the **start** and **accept** states



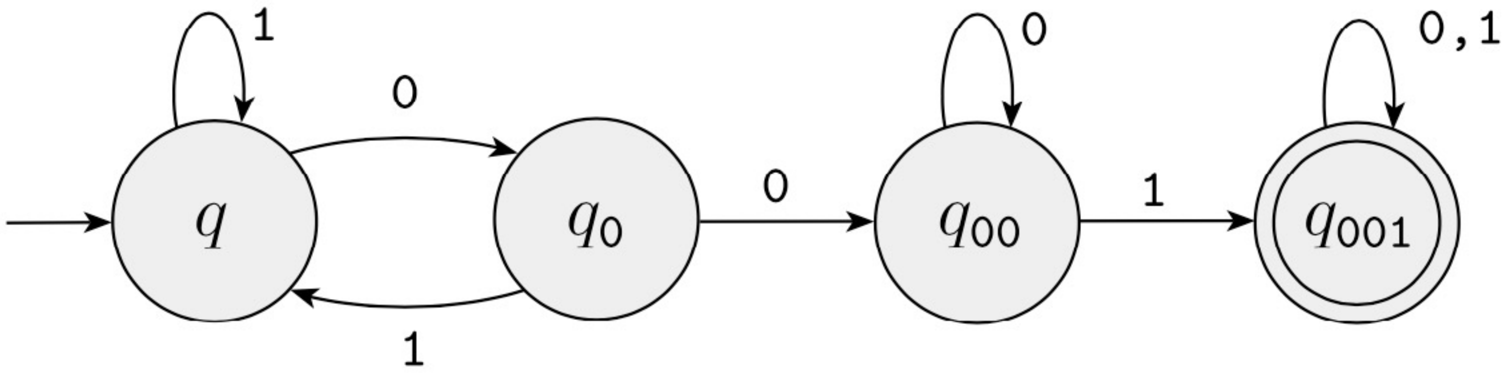
Example 2

- Design a finite automaton E_2 to recognize the regular language of all strings that contain the string 001 as a substring.
  - For example, 0010, 1001, 001, 1111001111 are all in the language, but 00 and 11 are not.

Example 2

- There are four possibilities to keep track of:
  1. Haven't just seen any symbols of the pattern,
  2. Have just seen a 0,
  3. Have just seen a 00, or
  4. Have seen the entire pattern 001.

Example 2



The regular operations

- In arithmetic, the basic objects are *numbers* and the tools for manipulating them are *arithmetic operations* such addition and multiplication.

- Analogously, in theory of computation, the objects are *languages* and the tools for manipulating them include *regular operations*.

- Regular operations are used to study *properties of regular languages*.

The regular operations

DEFINITION 1.23

Let A and B be languages. We define the regular operations ***union***, ***concatenation***, and ***star*** as follows:

- **Union**: $A \cup B = \{x | \, x \in A \text{ or } x \in B\}$.
- **Concatenation**: $A \circ B = \{xy | \, x \in A \text{ and } y \in B\}$.
- **Star**: $A^* = \{x_1 x_2 \ldots x_k | \, k \geq 0 \text{ and each } x_i \in A\}$.

Example

Let the alphabet Σ be the standard 26 letters $\{a,b,\ldots,z\}$.

If A = {cool, boring} and B = {student, teacher}, then

- $A \cup B$ = {cool, boring, student, teacher}
- $A \circ B$ = {coolstudent, coolteacher, boringstudent, boringteacher}, and
- A^* = {ε, cool, boring, coolcool, coolboring, boringcool, boringboring, coolcoolcool, coolcoolboring, coolboringcool, ….}

Closed under….

- Let $N = \{1,2,3,\ldots\}$ be the set of natural numbers.
- We say that N is *closed under multiplication,* because for any x and y in N, the product $x \times y$ is also in N.
- In contrast, N is not closed under division, as 2 and 3 are in N but 2/3 is not.

- Generally, a collection of objects is closed under some operation if applying that operation to members of the collection return an object still in the collection.

- *We will show that the collection of regular languages is closed under all three of the regular operations*.

Theorem

The class of regular languages is closed under the union operation

IOW: if A_1 and A_2 are regular languages, so is $A \cup B$.

Proof Idea

- Have regular languages A_1 and A_2 and want to show that A_1 U A_2 also is regular
- Because A_1 and A_2 are regular, we know that some FA M_1 recognizes A_1 and some FA M_2 recognizes A_2.
- To prove that A_1 U A_2 is regular, we demonstrate a FA M that recognizes A_1 U A_2.

- The proof is by *construction*. We construct M from M_1 and M_2.
- M must accept its input exactly when either M_1 or M_2 would accept it in order to recognize the union language.
- It works by **simulating** both M_1 and M_2 and accepting if either of the simulations accept.
- First approach: simulate first M_1 and then simulate M_2.
  - Doesn't work since we can't rewind the input tape
- An approach that works: remember ***pairs*** of states

Formal proof

Let M_1 recognize A_1, where $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, and
M_2 recognize A_2, where $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.

Construct M to recognize $A_1 \cup A_2$, where $M = (Q, \Sigma, \delta, q_0, F)$.

1. $Q = \{(r_1, r_2) | \; r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$.
   This set is the ***Cartesian product*** of sets Q_1 and Q_2 and is written $Q_1 \times Q_2$. It is the set of all pairs of states, the first from Q_1 and the second from Q_2.
2. Σ, the alphabet, is the same as in M_1 and M_2. In this theorem and in all subsequent similar theorems, we assume for simplicity that both M_1 and M_2 have the same input alphabet Σ. The theorem remains true if they have different alphabets, Σ_1 and Σ_2. We would then modify the proof to let $\Sigma = \Sigma_1 \cup \Sigma_2$.
3. δ, the transition function, is defined as follows. For each $(r_1, r_2) \in Q$ and each $a \in \Sigma$, let
   $$\delta\big((r_1, r_2), a\big) = \big(\delta_1(r_1, a), \delta_2(r_2, a)\big).$$
   Hence δ gets a state of M (which actually is a pair of states from M_1 and M_2), together with an input symbol, and returns M's next state.
4. q_0 is the pair (q_1, q_2).
5. F is the set of pairs in which either member is an accept state of M_1 or M_2. We can write it as
   $$F = \{(r_1, r_2) | \; r_1 \in F_1 \text{ or } r_2 \in F_2\}.$$

Theorem

The class of regular languages is closed under the concatenation operation

IOW: if A_1 and A_2 are regular languages, so is $A \circ B$.

Proof attempt

- Start with FA M_1 and M_2 recognizing the regular languages A_1 and A_2.
- Construct a FA M that must accept an input if it can be broken into two pieces, where M_1 accepts the first piece and M_2 accepts the second piece.
- Problem: M doesn't know know where to break the input.

$$w = s_1 s_2 s_3 \cdots s_n$$

- We need a different strategy – nondeterminism – the subject of next lecture!