

# Harvard CS 121 and CSCI E-121

## Lecture 3: Finite Automata

Harry Lewis

September 10, 2013

**Reading:** Sipser, §1.1 and §1.2.

# Languages

- A **language**  $L$  over alphabet  $\Sigma$  is a set of strings over  $\Sigma$  (i.e.  $L \subseteq \Sigma^*$ )

Computational problem: given  $x \in \Sigma^*$ , is  $x \in L$ ?

Any YES/NO problems can be cast as a language.

- Examples of simple languages:
  - All words in the *American Heritage Dictionary*  $\{a, aah, aardvark, \dots, zyzzyva\}$ .
  - $\emptyset$
  - $\Sigma^*$
  - $\Sigma$
  - $\{x \in \Sigma^* : |x| = 3\} = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$

## More complicated languages

- The set of strings  $x \in \{a, b\}^*$  such that  $x$  has more  $a$ 's than  $b$ 's.
- The set of strings  $x \in \{0, 1\}^*$  such that  $x$  is the binary representation of a prime number.
- All 'C' programs that do not go into an infinite loop.
- $L_1 \cup L_2$ ,  $L_1 \cap L_2$ ,  $L_1 - L_2$  if  $L_1$  and  $L_2$  are languages.
- $\vdots$

## The highly abstract and metaphorical term “language”

- A language can be either finite or infinite
- A language need not have any “internal structure”

## Be careful to distinguish

$\varepsilon$  The empty string (a string)

$\emptyset$  The empty set (a set, possibly a language)

$\{\varepsilon\}$  The set containing one element, which is the empty string (a language)

$\{\emptyset\}$  The set containing one element, which is the empty set (a set of sets, maybe a set of languages)

# Operations on Languages

- Set operations  $\cup$   $\cap$   $-$

- Concatenation of Languages

$$L_1L_2 = \{xy : x \in L_1, y \in L_2\}$$

e.g.  $\{a, b\}\{a, bb\} = \{aa, ba, abb, bbb\}$

e.g.  $\{\varepsilon\}L = L$

e.g.  $\emptyset L = ?$

## Kleene star

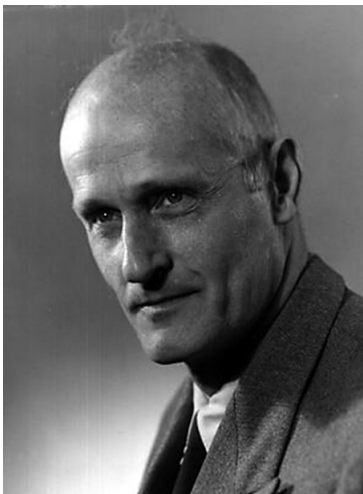
$$L^* = \{w_1 \cdots w_n : n \geq 0, w_1, \dots, w_n \in L\}$$

e.g.  $\{aa\}^* = \{\varepsilon, aa, aaaa, \dots\}$

e.g.  $\{ab, ba, aa, bb\}^* = \text{all even length strings}$

e.g.  $\Sigma^* = \text{Kleene Star of } \Sigma$

e.g.  $\emptyset^* = ?$



## (Deterministic) Finite Automata

### Example: Home Stereo

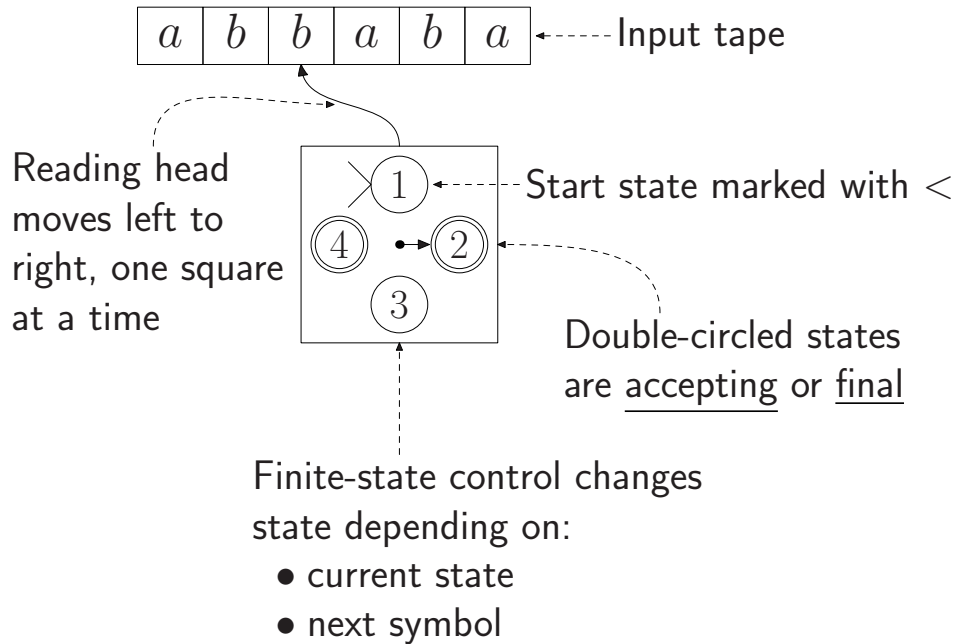
- $P$  = power button (ON/OFF)
- $S$  = source button (CD/Radio/TV), only works when stereo is ON, but source remembered when stereo is OFF.
- Starts OFF, in CD mode.
- A computational problem: does a given a sequence of button presses  $w \in \{P, S\}^*$  leave the system with the radio on?
- Does this really model the situation adequately?
  - This abstraction rules out pushing 2 buttons at once, holding one down while pushing the other, etc.

# The Home Stereo DFA

## Formal Definition of a DFA

- A DFA  $M$  is a 5-Tuple  $(Q, \Sigma, \delta, q_0, F)$ 
  - $Q$  : Finite set of states
  - $\Sigma$  : Alphabet
  - $\delta$  : “Transition function”,  $Q \times \Sigma \rightarrow Q$
  - $q_0$  : Start state,  $q_0 \in Q$
  - $F$  : Accept (or final) states,  $F \subseteq Q$
- If  $\delta(p, \sigma) = q$ ,
  - then if  $M$  is in state  $p$  and reads symbol  $\sigma \in \Sigma$
  - then  $M$  enters state  $q$  (while moving to next input symbol)
- Home Stereo example:

## Another Visualization



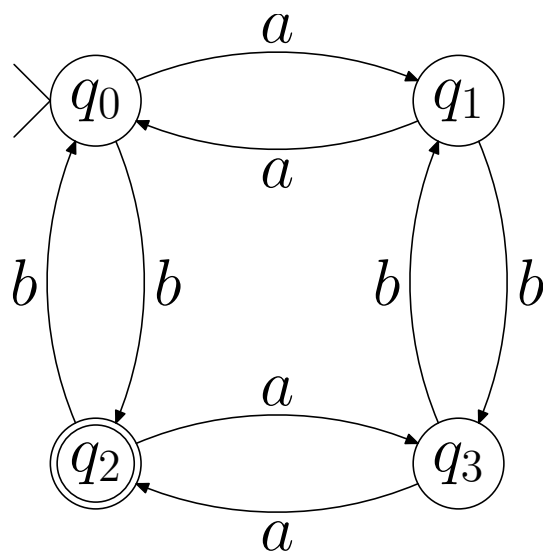
$M$  accepts string  $x$  if

- After starting  $M$  in the start[initial] state with head on first square,
- when all of  $x$  has been read,
- $M$  winds up in a final state.

## Examples

- Bounded Counting: A DFA for

$\{x : x \text{ has an even \# of } a\text{'s and an odd \# of } b\text{'s}\}$



Transition  
function  $\delta$ :

	$a$	$b$
$q_0$	$q_1$	$q_2$
$q_1$	$q_0$	$q_3$
$q_2$	$q_3$	$q_0$
$q_3$	$q_2$	$q_1$

i.e.  
 $\delta(q_0, a) =$   
 $q_1$ , etc.

 = start state       = final state

$Q = \{q_0, q_1, q_2, q_3\}$      $\Sigma = \{a, b\}$      $F = \{q_2\}$

## Another Example, to work out together

- Pattern Recognition: A DFA that accepts  $\{ x : x \text{ has } aab \text{ as a substring} \}$ .

## Another Example

- Pattern Recognition: A DFA that accepts  $\{ x : x \text{ has } ababa \text{ as a substring} \}$ .

## Another Example

- A DFA that accepts  $\{ x : x \text{ has } ababa \text{ as a substring} \}$ .

You are going through a constructive process

string  $\rightarrow$  DFA

that is automated in every text editor!

Really a compiler that generates DFA code from an input string pattern

## **DFAs are very simple but a great deal is still unknown!**

How big is the smallest automaton that can tell two given strings apart?

## DFAs are very simple but a great deal is still unknown!

How big is the smallest automaton that can tell two given strings apart?

That is, for any strings  $x, y$ , let  $S(x, y)$  be the number of states in the smallest DFA  $M$  that accepts  $x$  and rejects  $y$ . (Doesn't matter what  $M$  does when given other strings as input.)

For  $n \geq 0$ , what is  $S(n) = \max\{S(x, y) : |x|, |y| \leq n\}$ ?

## Formal Definition of Computation

$M = (Q, \Sigma, \delta, q_0, F)$  accepts  $w = w_1w_2 \cdots w_n \in \Sigma^*$  (where each  $w_i \in \Sigma$ ) if there exist  $r_0, \dots, r_n \in Q$  such that

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

The language recognized (or accepted) by  $M$ , denoted  $L(M)$ , is the set of all strings accepted by  $M$ .

**Example:**

## Transition function on an entire string

More formal (not necessary for us, but notation sometimes useful):

- Inductively define  $\delta^* : Q \times \Sigma^* \rightarrow Q$  by  $\delta^*(q, \varepsilon) = q$ ,  
 $\delta^*(q, w\sigma) = \delta(\delta^*(q, w), \sigma)$ .
- Intuitively,  $\delta^*(q, w) =$   
“state reached after starting in  $q$  and reading the string  $w$ ”.
- $M$  accepts  $w$  if  $\delta^*(q_0, w) \in F$ .

## Transition function on an entire string

More formal (not necessary for us, but notation sometimes useful):

- Inductively define  $\delta^* : Q \times \Sigma^* \rightarrow Q$  by  $\delta^*(q, \varepsilon) = q$ ,  
 $\delta^*(q, w\sigma) = \delta(\delta^*(q, w), \sigma)$ .
- Intuitively,  $\delta^*(q, w) =$   
“state reached after starting in  $q$  and reading the string  $w$ ”.
- $M$  accepts  $w$  if  $\delta^*(q_0, w) \in F$ .

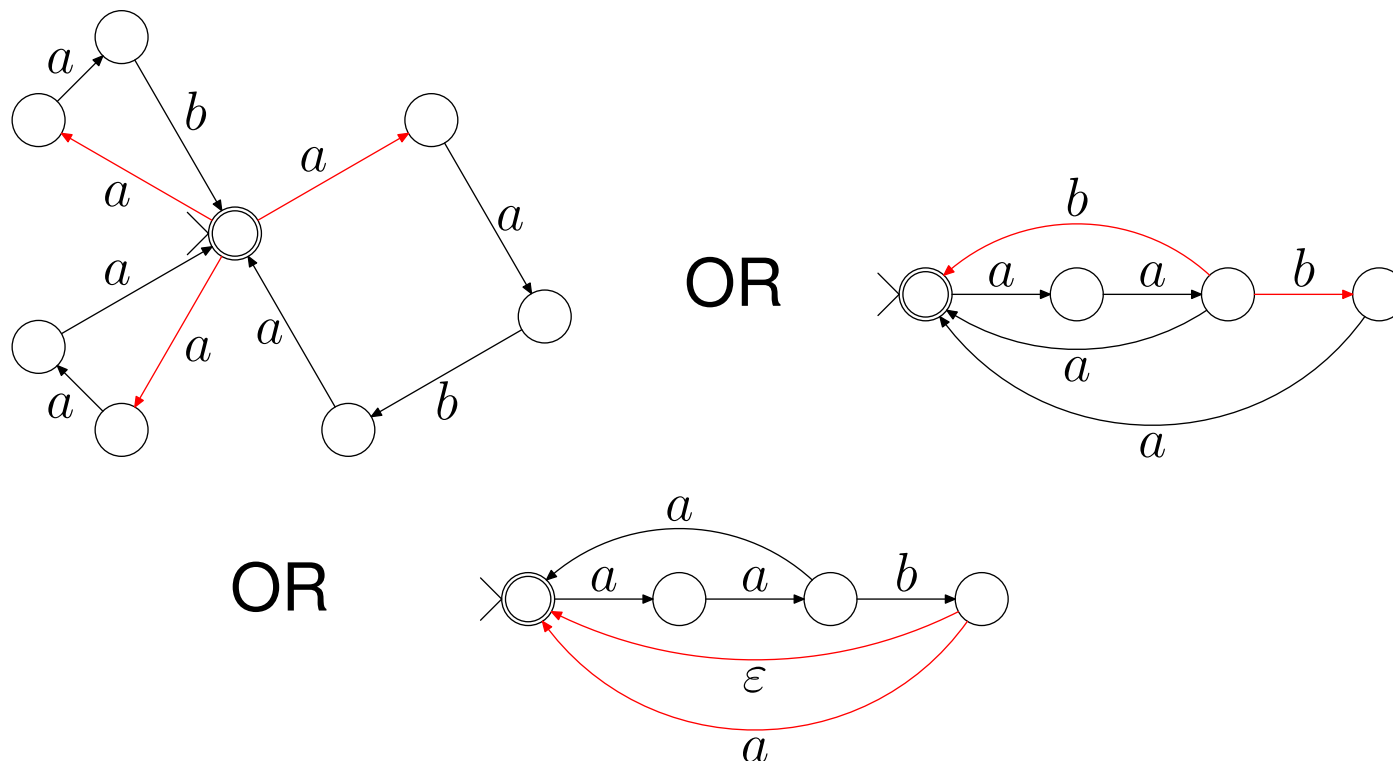
**Determinism:** Given  $M$  and  $w$ , the states  $r_0, \dots, r_n$  are uniquely determined. Or in other words,  $\delta^*(q, w)$  is well defined for any  $q$  and  $w$ : There is precisely one state to which  $w$  “drives”  $M$  if it is started in a given state.

## The impulse for nondeterminism

A language for which it is hard to design a DFA:

$$\{x_1x_2 \cdots x_k : k \geq 0 \text{ and each } x_i \in \{aab, aaba, aaa\}\}.$$

But it is easy to imagine a “device” to accept this language if there sometimes can be several possible transitions!



## Nondeterministic Finite Automata

An NFA is a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$ , where

- $Q, \Sigma, q_0, F$  are as for DFAs
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow P(Q)$ .

When in state  $p$  reading symbol  $\sigma$ , can go to any state  $q$  in the set  $\delta(p, \sigma)$ .

- there may be more than one such  $q$ , or
- there may be none (in case  $\delta(p, \sigma) = \emptyset$ ).

Can “jump” from  $p$  to any state in  $\delta(p, \varepsilon)$  without moving the input head.

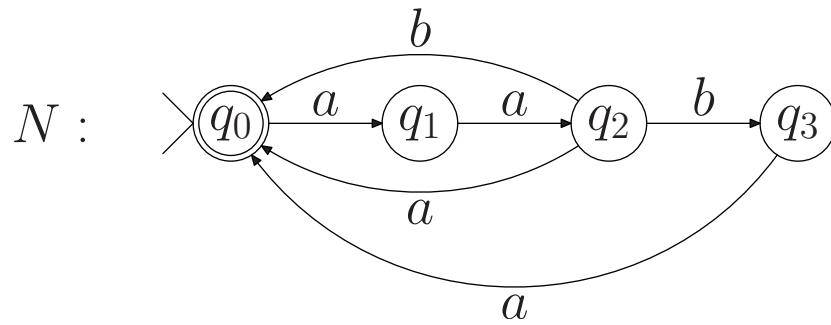
## Computations by an NFA

$N = (Q, \Sigma, \delta, q_0, F)$  accepts  $w \in \Sigma^*$  if we can write  $w = y_1 y_2 \cdots y_m$  where each  $y_i \in \Sigma \cup \{\varepsilon\}$  and there exist  $r_0, \dots, r_m \in Q$  such that

1.  $r_0 = q_0$ ,
2.  $r_{i+1} \in \delta(r_i, y_{i+1})$  for each  $i = 0, \dots, m - 1$ , and
3.  $r_m \in F$ .

**Nondeterminism:** Given  $N$  and  $w$ , the states  $r_0, \dots, r_m$  are not necessarily determined.

## Example of an NFA



$N = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \delta, q_0, \{q_0\})$ , where  $\delta$  is given by:

	$a$	$b$	$\varepsilon$
$q_0$	$\{q_1\}$	$\emptyset$	$\emptyset$
$q_1$	$\{q_2\}$	$\emptyset$	$\emptyset$
$q_2$	$\{q_0\}$	$\{q_0, q_3\}$	$\emptyset$
$q_3$	$\{q_0\}$	$\emptyset$	$\emptyset$

Work out the tree of all possible computations on  $aabaab$

## Tree of computations

Tree of computations of NFA  $N$  on string  $aabaab$ :

## How to simulate NFAs?

- NFA accepts  $w$  if there is at least one accepting computational path on input  $w$
- But the number of paths may grow exponentially with the length of  $w$ !
- Can exponential search be avoided?

## NFAs vs. DFAs

NFAs seem more “powerful” than DFAs. Are they?