

# Computational Theory

## Finite Automata and Regular Languages

Dr Russ Ross

Dixie State University—Computer and Information Technologies

Fall 2017

Adapted from notes by Harry Lewis

# Finite Automata

**Reading:** Sipser §1.1 and §1.2.

# Deterministic Finite Automata (DFAs)

## 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 sequence of button presses  $w \in \{P, S\}^*$  leave the system with the radio on?

# 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

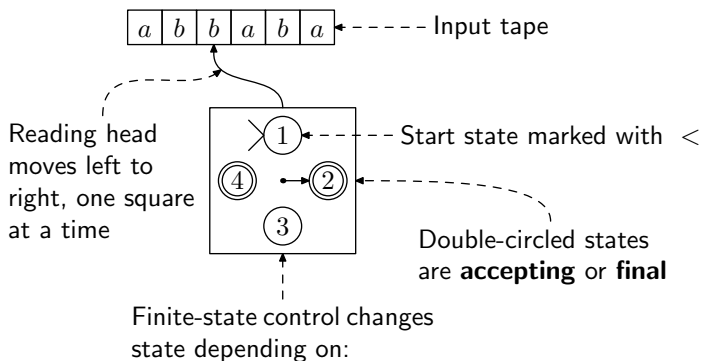
$\delta$  : “Transition function”,  $Q \times \Sigma \rightarrow Q$

$q_0$  : 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



- current state
- next symbol

# Accepting Strings

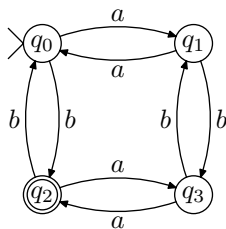
$M$  accepts string  $X$  if

- ▶ 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

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

## Another Example, to work out together

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



# Formal Definition of Computation

$M = (Q, \Sigma, \delta, q_0, F)$  **accepts**  $w = w_1 w_2 \cdots w_n \in \Sigma^*$   
(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  $M$ .

# 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  $\delta^*(q_0, w) \in F$ .

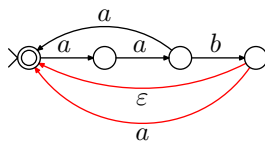
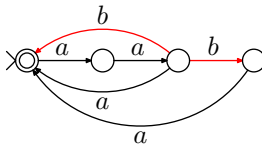
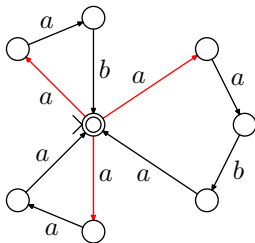
**Determinism:** Given  $M$  and  $w$ , the states  $r_0, \dots, r_n$  are uniquely determined. Or in other words,  $\delta^*(q, w)$  is well defined for any  $q$  and  $w$ : There is precisely one state to which  $w$  “drives”  $M$  if it is started in a given state.

# The impulse for nondeterminism

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

$$\{x_1 x_2 \cdots x_k : k \geq 0 \text{ and each } x_i \in \{aab, aaba, aaa\}\}$$

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
- ▶  $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q)$

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.

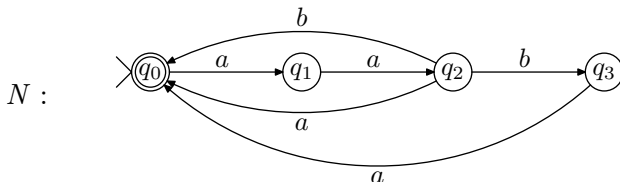
# Computations by an NFA

$N = (Q, \Sigma, \delta, q_0, F)$  **accepts**  $w \in \Sigma^*$  if we can write  $w = y_1 y_2 \dots y_m$  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  $aabaab$

# 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 and DFAs Closure Properties

**Reading:** Sipser §1.2.

# NFAs vs. DFAs

NFAs *seem* more powerful than DFAs. Are they?

**Theorem:** For every NFA  $N$ , there exists a DFA  $M$  such that  $L(M) = L(N)$ .

**Proof Outline:** Given any NFA  $N$ , to construct a DFA  $M$  such that  $L(M) = L(N)$ :

# NFAs vs. DFAs

NFAs *seem* more powerful than DFAs. Are they?

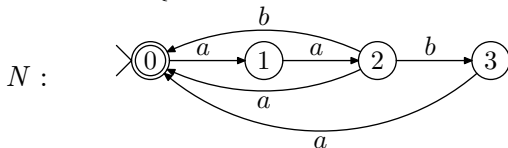
**Theorem:** For every NFA  $N$ , there exists a DFA  $M$  such that  $L(M) = L(N)$ .

**Proof Outline:** Given any NFA  $N$ , to construct a DFA  $M$  such that  $L(M) = L(N)$ :

- ▶ Have the DFA keep track, at all times, of all possible states the NFA could be in after reading the same initial part of the input string.
- ▶ I.e., the **states** of  $M$  are **sets** of states of  $N$ , and  $\delta_M^*(R, w)$  is the set of all states  $N$  could reach after reading  $w$ , starting from a state in  $R$ .

# Example of the SUBSET CONSTRUCTION

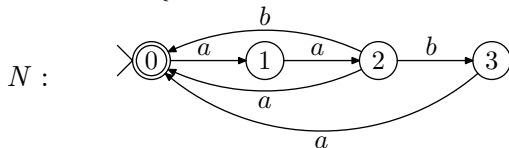
NFA  $N$  for  $\{x_1x_2\cdots x_k : k \geq 0 \text{ and each } x_i \in \{aab, aaba, aaa\}\}$ .



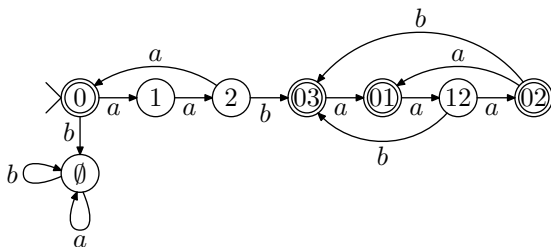
$N$  starts in state 0 so we will construct a DFA  $M$  starting in state  $\{0\}$ .

# Example of the SUBSET CONSTRUCTION

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



$N$  starts in state 0 so we will construct a DFA  $M$  starting in state  $\{0\}$ . Here it is:



All other transitions are to the “dead state”  $\emptyset$ . The other states are unreachable, though technically must be defined. Final states are all those containing 0, the final state of  $N$ .

# Formal Construction of DFA $M$ from NFA $N = (Q, \Sigma, \delta, q_0, F)$

On the assumption that  $\delta(p, \varepsilon) = \emptyset$  for all states  $p$ .

(i.e., we assume no  $\varepsilon$ -transitions, just to simplify things a bit)

$M = (Q', \Sigma, \delta', q'_0, F')$  where

$$\begin{aligned}Q' &= \mathcal{P}(Q) \\q'_0 &= \{q_0\} \\F' &= \{R \subseteq Q : R \cap F \neq \emptyset\} \text{ (that is, } R \in Q') \\ \delta'(R, \sigma) &= \{q \in Q : q \in \delta(r, \sigma) \text{ for some } r \in R\} \\ &= \bigcup_{r \in R} \delta(r, \sigma)\end{aligned}$$

# Proving that the construction works

**Claim:** For every string  $w$ , running  $M$  on input  $w$  ends in the state  $\{q \in Q : \text{some computation of } N \text{ on input } w \text{ ends in state } q\}$ .

**Pf:** By induction on  $|w|$ .

Can be extended to work even for NFAs with  $\varepsilon$ -transitions.

“THE SUBSET CONSTRUCTION”

# Closure Properties

**Theorem:** The class of regular languages is closed under:

- ▶ Union:  $L_1 \cup L_2$
- ▶ Concatenation:  $L_1 \circ L_2 = \{xy : x \in L_1 \text{ and } y \in L_2\}$
- ▶ Kleene \*:  $L_1^* = \{x_1x_2 \cdots x_k : k \geq 0 \text{ and each } x_i \in L_1\}$
- ▶ Complement:  $\overline{L_1}$
- ▶ Intersection:  $L_1 \cap L_2$

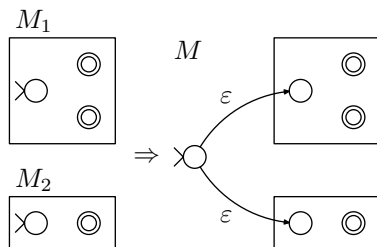


# Closure Properties

**Theorem:** The class of regular languages is closed under:

- ▶ Union:  $L_1 \cup L_2$
- ▶ Concatenation:  $L_1 \circ L_2 = \{xy : x \in L_1 \text{ and } y \in L_2\}$
- ▶ Kleene \*:  $L_1^* = \{x_1x_2 \cdots x_k : k \geq 0 \text{ and each } x_i \in L_1\}$
- ▶ Complement:  $\overline{L_1}$
- ▶ Intersection:  $L_1 \cap L_2$

**Union:** If  $L_1$  and  $L_2$  are regular, then  $L_1 \cup L_2$  is regular.



$M$  has the states and transitions of  $M_1$  and  $M_2$  plus a new start state  $\epsilon$ -transitioning to the old start states.

# Concatenation, Kleene\*, Complementation

## Concatenation:

$$L(M) = L(M_1) \circ L(M_2)$$

## Kleene\*:

$$L(M) = L(M_1)^*$$

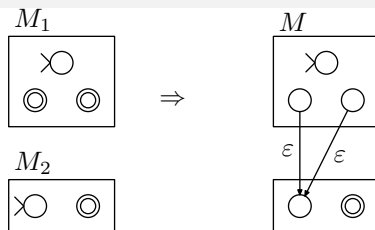
## Complement:

$$L(M) = \overline{L(M_1)}$$

# Concatenation, Kleene\*, Complementation

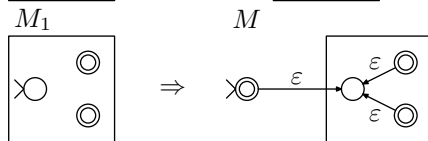
## Concatenation:

$$L(M) = L(M_1) \circ L(M_2)$$



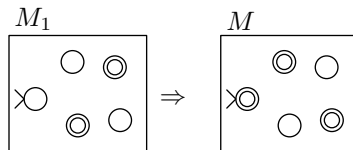
## Kleene\*:

$$L(M) = L(M_1)^*$$



## Complement:

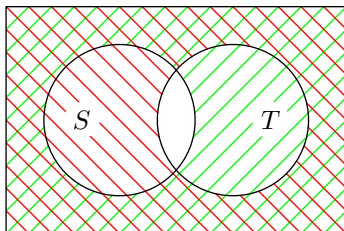
$$L(M) = \overline{L(M_1)}$$





- ▶ Assume  $M$  is deterministic (or make it so)
- ▶ Invert final/nonfinal states

# Closure under intersection

**Intersection:**  $S \cap T = \overline{\overline{S} \cup \overline{T}}$



  $= \overline{S}$

  $= \overline{T}$

Hence closure under union  
and complement implies  
closure under intersection.

# A more constructive and direct proof of closure under intersection

Better way (“Cross Product Construction”):

From DFAs  $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$  and  $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ , construct  $M = (Q, \Sigma, \delta, q_0, F)$ :

$$\begin{aligned}Q &= Q_1 \times Q_2 \\F &= F_1 \times F_2 \\ \delta(\langle r_1, r_2 \rangle, \sigma) &= \langle \delta_1(r_1, \sigma), \delta_2(r_2, \sigma) \rangle \\ q_0 &= \langle q_1, q_2 \rangle\end{aligned}$$

Then  $L(M_1) \cap L(M_2) = L(M)$

# Some Efficiency Considerations

The subset construction shows that any  $n$ -state NFA can be implemented as a  $2^n$ -state DFA.

NFA States	DFA States
4	16
10	1024
100	$2^{100}$
1000	$2^{1000} \gg$ the number of particles in the universe

How to implement this construction on an ordinary digital computer?

NFA states

$1, \dots, n$

DFA state bit vector

0	1	1	0	...	1
1	2				$n$

# Is this construction the best we can do?

Could there be a construction that always produces an  $n^2$  state DFA for example?

**Theorem:** For every  $n \geq 1$ , there is a language  $L_n$  such that

1. There is an  $(n + 1)$ -state NFA recognizing  $L_n$ .
2. There is no DFA recognizing  $L_n$  with fewer than  $2^n$  states.

**Conclusion:** For finite automata, nondeterminism provides an **exponential savings** over determinism (in the worst case).

# Proving that exponential blowup is sometimes unavoidable

(Could there be a construction that always produces an  $n^2$  state DFA for example?)

Consider (for some fixed  $n = 17$ , say)

$L_n = \{w \in \{a, b\}^* : \text{the } n\text{th symbol from the right end of } w \text{ is an } a\}$

- ▶ There is an  $(n + 1)$ -state NFA that accepts  $L_n$ .
- ▶ There is no DFA that accepts  $L_n$  and has  $< 2^n$  states

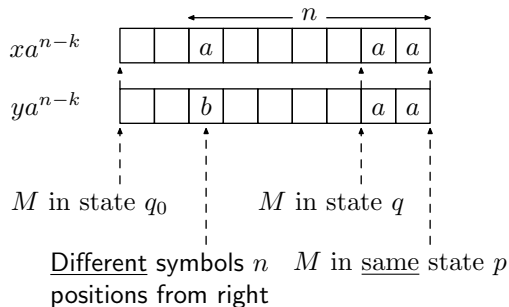
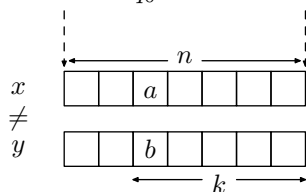


# A “Fooling Argument”

- ▶ Suppose a DFA  $M$  has  $< 2^n$  states, and  $L(M) = L_n$
- ▶ There are  $2^n$  strings of length  $n$ .
- ▶ By the pigeonhole principle, two such strings  $x \neq y$  must drive  $M$  to the same state  $q$ .
- ▶ Suppose  $x$  and  $y$  differ at the  $k^{th}$  position from the right end (one has  $a$ , the other has  $b$ )  
( $k = 1, 2, \dots$ , or  $n$ )
- ▶ Then  $M$  must treat  $xa^{n-k}$  and  $ya^{n-k}$  identically (accept both or reject both). These strings differ at position  $n$  from the right end.
- ▶ So  $L(M) \neq L_n$ , contradiction. QED.

# Illustration of the fooling argument

$M$  is in state  $q_0$        $M$  is in state  $q$



- ▶  $x$  and  $y$  are different strings  
(so there is a position  $k$  where one has  $a$  and the other has  $b$ )
- ▶ But both strings drive  $M$  from  $s$  to the same state  $q$

# What the argument proves

- ▶ This shows that the subset construction is within a factor of 2 of being optimal
- ▶ In fact it is optimal, i.e., as good as we can do in the worst case
- ▶ In many cases, the “generate-states-as-needed” method yields a DFA with  $\ll 2^n$  states  
(e.g. if the NFA was deterministic to begin with!)

# Regular Expressions

**Reading:** Sipser §1.3.

# Regular Expressions

- ▶ Let  $\Sigma = \{a, b\}$ . The **regular expressions** over  $\Sigma$  are certain expressions formed using the symbols  $\{a, b, (, ), \varepsilon, \emptyset, \cup, \circ, *\}$
- ▶ We use **red** for the strings under discussion (the **object language**) and **black** for the ordinary notation we are using for doing mathematics (the **metalanguage**).
- ▶ **Construction Rules** (= inductive/recursive definition):
  1.  $a, b, \varepsilon, \emptyset$  are regular expressions
  2. If  $R_1$  and  $R_2$  are RE's, then so are  $(R_1 \circ R_2)$ ,  $(R_1 \cup R_2)$ , and  $(R_1^*)$ .
- ▶ Examples:
  - ▶  $(a \circ b)$
  - ▶  $((((a \circ (b^*)) \circ c) \cup ((b^*) \circ a))^*)$
  - ▶  $(\emptyset^*)$

# What REs Do

- ▶ Regular expressions (which are strings) represent languages (which are sets of strings), via the function  $L$ :

$$(1) \quad L(a) = \{a\}$$

$$(2) \quad L(b) = \{b\}$$

$$(3) \quad L(\varepsilon) = \{\varepsilon\}$$

$$(4) \quad L(\emptyset) = \emptyset$$

$$(5) \quad L((R_1 \circ R_2)) = L(R_1) \circ L(R_2)$$

$$(6) \quad L((R_1 \cup R_2)) = L(R_1) \cup L(R_2)$$

$$(7) \quad L((R_1^*)) = L(R_1)^*$$

- ▶ Example:

$$L(((a^*) \circ (b^*))) = \{a\}^* \circ \{b\}^*$$

- ▶  $L(\cdot)$  is called the **semantics** of the expression.

# Syntactic Shorthand

- ▶ Drop the distinction between red and black, between object language and metalanguage
- ▶ Omit  $\circ$  symbol and many parentheses
- ▶ Union and concatenation of languages are associative

i.e., for any languages  $L_1, L_2, L_3$ :

$$(L_1 L_2) L_3 = L_1 (L_2 L_3) \text{ and } (L_1 \cup L_2) \cup L_3 = L_1 \cup (L_2 \cup L_3)$$

so we can write just  $R_1 R_2 R_3$  and  $R_1 \cup R_2 \cup R_3$

For example, the following are all equivalent:

$$((ab)c) \quad (a(bc)) \quad abc$$

- ▶ **Equivalent** means “same semantics, maybe different syntax”

# More syntactic sugar

- ▶ By convention,  $*$  takes precedence over  $\circ$ , which takes precedence over  $\cup$ .

So  $a \cup bc^*$  is equivalent to  $(a \cup (b \circ (c^*)))$

- ▶  $\Sigma$  is shorthand for  $a \cup b$  (or the analogous RE for whatever alphabet is in use).



# Examples of Regular Languages

Strings ending in  $a = \Sigma^* a$

Strings containing the substring  $abaaab = ?$

Strings of even length  $= (aa \cup ab \cup ba \cup bb)^*$

Strings with even # of  $a$ 's  $= (b \cup ab^*a)^*$   
 $= b^*(ab^*ab^*)^*$

# Examples of Regular Languages

Strings ending in  $a = \Sigma^* a$

Strings containing the substring  $abaaab = ?$

Strings of even length  $= (aa \cup ab \cup ba \cup bb)^*$

Strings with even # of  $a$ 's  $= (b \cup ab^*a)^*$   
 $= b^*(ab^*ab^*)^*$

Strings with  $\leq$  two  $a$ 's  $= ?$

# Examples of Regular Languages

Strings ending in  $a = \Sigma^* a$

Strings containing the substring  $abaaab = ?$

Strings of even length  $= (aa \cup ab \cup ba \cup bb)^*$

Strings with even # of  $a$ 's  $= (b \cup ab^*a)^*$   
 $= b^*(ab^*ab^*)^*$

Strings with  $\leq$  two  $a$ 's  $= ?$

Strings of form  $x_1x_2 \dots x_k, k \geq 0$ , each  $x_i \in \{aab, aaba, aaa\} = ?$

# Examples of Regular Languages

Strings ending in  $a = \Sigma^* a$

Strings containing the substring  $abaaab = ?$

Strings of even length  $= (aa \cup ab \cup ba \cup bb)^*$

Strings with even # of  $a$ 's  $= (b \cup ab^*a)^*$   
 $= b^*(ab^*ab^*)^*$

Strings with  $\leq$  two  $a$ 's  $= ?$

Strings of form  $x_1x_2 \dots x_k, k \geq 0$ , each  $x_i \in \{aab, aaba, aaa\} = ?$

Decimal numerals, no leading zeros  
 $= 0 \cup ((1 \cup \dots \cup 9)(0 \cup \dots \cup 9)^*)$

# Examples of Regular Languages

Strings ending in  $a = \Sigma^* a$

Strings containing the substring  $abaaab = ?$

Strings of even length  $= (aa \cup ab \cup ba \cup bb)^*$

Strings with even # of  $a$ 's  $= (b \cup ab^*a)^*$   
 $= b^*(ab^*ab^*)^*$

Strings with  $\leq$  two  $a$ 's  $= ?$

Strings of form  $x_1x_2 \dots x_k, k \geq 0$ , each  $x_i \in \{aab, aaba, aaa\} = ?$

Decimal numerals, no leading zeros  
 $= 0 \cup ((1 \cup \dots \cup 9)(0 \cup \dots \cup 9)^*)$

All strings with an even # of  $a$ 's *and* an even # of  $b$ 's  
 $= (b \cup ab^*a)^* \cap (a \cup ba^*b)^*$  *but this isn't a regular expression*

# Examples of Regular Languages

Strings ending in  $a = \Sigma^* a$

Strings containing the substring  $abaaab = ?$

Strings of even length  $= (aa \cup ab \cup ba \cup bb)^*$

Strings with even # of  $a$ 's  $= (b \cup ab^*a)^*$   
 $= b^*(ab^*ab^*)^*$

Strings with  $\leq$  two  $a$ 's  $= ?$

Strings of form  $x_1x_2 \dots x_k, k \geq 0$ , each  $x_i \in \{aab, aaba, aaa\} = ?$

Decimal numerals, no leading zeros  
 $= 0 \cup ((1 \cup \dots \cup 9)(0 \cup \dots \cup 9)^*)$

All strings with an even # of  $a$ 's *and* an even # of  $b$ 's  
 $= (b \cup ab^*a)^* \cap (a \cup ba^*b)^*$  *but this isn't a regular expression*  
 $= (aa \cup bb)^*((ab \cup ba)(aa \cup bb)^*(ab \cup ba)(aa \cup bb)^*)^*$

# Equivalence of REs and FAs

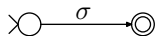
Recall: we call a language **regular** if there is a finite automaton that recognizes it.

**Theorem:** For every regular expression  $R$ ,  $L(R)$  is regular.

**Proof** (going back to hyper-formality for a moment):

Induct on the construction of regular expressions (“structural induction”).

**Base Case:**  $R$  is  $a$ ,  $b$ ,  $\varepsilon$ , or  $\emptyset$



accepts  $\{\sigma\}$



accepts  $\emptyset$



accepts  $\{\varepsilon\}$

## Equivalence of REs and FAs, continued

**Inductive Step:** If  $R_1$  and  $R_2$  are REs and  $L(R_1)$  and  $L(R_2)$  are regular (inductive hyp.), then so are:

$$L((R_1 \circ R_2)) = L(R_1) \circ L(R_2)$$

$$L((R_1 \cup R_2)) = L(R_1) \cup L(R_2)$$

$$L((R_1^*)) = L(R_1)^*$$

(By the closure properties of the regular languages).

Proof is **constructive** (actually produces the equivalent NFA, not just proves its existence).



# Example conversion of a RE to a FA

$$(a \cup \varepsilon)(aa \cup bb)^*$$

# The Other Direction

**Theorem:** For every regular language  $L$ , there is a regular expression  $R$  such that  $L(R) = L$ .

**Proof:**

Define **generalized NFAs** (GNFAs) (of interest only for this proof)

- ▶ Transitions labelled by regular expressions (rather than symbols).
- ▶ One start state  $q_{\text{start}}$  and only one accept state  $q_{\text{accept}}$ .
- ▶ Exactly one transition from  $q_i$  to  $q_j$  for every two states  $q_i \neq q_{\text{accept}}$  and  $q_j \neq q_{\text{start}}$  (including self-loops).

# Steps toward the proof

**Lemma:** For every NFA  $N$ , there is an equivalent GNFA  $G$ .

- ▶ Add new start state, new accept state. Transitions?
- ▶ If multiple transitions between two states, combine. How?
- ▶ If no transition between two states, add one. With what transition?

**Lemma:** For every GNFA  $G$ , there is an equivalent RE  $R$ .

- ▶ By induction on the number of states  $k$  of  $G$ .
- ▶ **Base case:**  $k = 2$ . Set  $R$  to be the label of the transition from  $q_{\text{start}}$  to  $q_{\text{accept}}$ .

# Ripping and repairing GNFA's to reduce the number of states

- ▶ **Inductive Hypothesis:** Suppose every GNFA  $G$  of  $k$  or fewer states has an equivalent RE (where  $k \geq 2$ ).
- ▶ **Induction Step:** Given a  $(k + 1)$ -state GNFA  $G$ , we will construct an equivalent  $k$ -state GNFA  $G'$ .

**Rip:** Remove a state  $q_r$  (other than  $q_{\text{start}}$ ,  $q_{\text{accept}}$ ).

**Repair:** For every two states  $q_i \notin \{q_{\text{accept}}, q_r\}$ ,  $q_j \notin \{q_{\text{start}}, q_r\}$ , let  $R_{i,j}$ ,  $R_{i,r}$ ,  $R_{r,r}$ ,  $R_{r,j}$  be REs on transitions  $q_i \rightarrow q_j$ ,  $q_i \rightarrow q_r$ ,  $q_r \rightarrow q_r$  and  $q_r \rightarrow q_j$  in  $G$ , respectively,

In  $G'$ , put RE  $R_{i,j} \cup R_{i,r}R_{r,r}^*R_{r,j}$  on transition  $q_i \rightarrow q_j$ .

Argue that  $L(G') = L(G)$ , which is regular by IH.

Also **constructive**.

## Example conversion of an NFA to a RE

An NFA accepting strings with an even number of  $a$ 's and an even number of  $b$ 's.

# Countability

**Reading:** Sipser, “The Diagonalization Method,” pages 174–178 (from just before Definition 4.12 up to Corollary 4.18).

# Examples of Regular Languages

- ▶  $\{w \in \{a, b\}^* : |w| \text{ even \& every 3rd symbol is an } a\}$
- ▶  $\{w \in \{a, b\}^* : \text{There are not 7 } a\text{'s or 7 } b\text{'s in a row}\}$
- ▶  $\{w \in \{a, b\}^* : w \text{ has both an even number of } a\text{'s and an even number of } b\text{'s}\}$
- ▶  $\{w : w \text{ is written using the ASCII character set and every substring delimited by spaces, punctuation marks, or the beginning or end of the string is in the American Heritage Dictionary}\}$

# Questions about regular languages

Give  $X$  = a regular expression, DFA, or NFA, how could you tell if:

- ▶  $x \in L(X)$ , where  $x$  is some string?
- ▶  $L(X) = \emptyset$ ?
- ▶  $x \in L(X)$  but  $x \notin L(Y)$ ?
- ▶  $L(X) = L(Y)$ , where  $Y$  is another RE/FA?
- ▶  $L(X)$  is infinite?
- ▶ There are infinitely many strings that belong to both  $L(X)$  and  $L(Y)$ ?



# Goal: Existence of Non-Regular Languages

Intuition:

- ▶ Every regular language can be described by a finite string (namely a regular expression).
- ▶ To specify an arbitrary language requires an infinite amount of information.
  - ▶ For example, an infinite sequence of bits would suffice.
  - ▶  $\Sigma^*$  has a lexicographic ordering, and the  $i$ 'th bit of an infinite sequence specifying a language would say whether or not the  $i$ 'th string is in the language.

⇒ Some languages must not be regular.

How to formalize?

# Countability

- ▶ A set  $S$  is **finite** if there is a bijection  $\{1, \dots, n\} \leftrightarrow S$  for some  $n \geq 0$ .
- ▶ **Countably infinite** if there is a bijection  $f : \mathcal{N} \leftrightarrow S$

This means that  $S$  can be “enumerated,” i.e. listed as  $\{s_0, s_1, s_2, \dots\}$  where  $s_i = f(i)$  for  $i = 0, 1, 2, 3, \dots$

So  $\mathcal{N}$  itself is countably infinite

So is  $\mathcal{Z}$  (integers) since  $\mathcal{Z} = \{0, -1, 1, -2, 2, \dots\}$

Q: What is  $f$ ?

- ▶ **Countable** if  $S$  is finite or countably infinite
- ▶ **Uncountable** if it is not countable

# Facts about Infinite Sets

- **Proposition:** The union of 2 countably infinite sets is countably infinite.

$$\text{If } A = \{a_0, a_1, \dots\}, B = \{b_0, b_1, \dots\}$$

$$\text{The } A \cup B = C = \{c_0, c_1, \dots\}$$

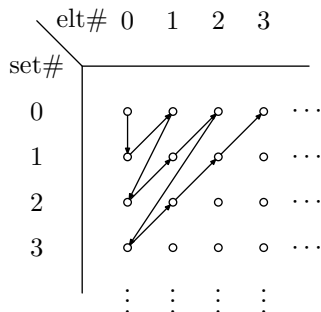
$$\text{where } c_i = \begin{cases} a_{i/2} & \text{if } i \text{ is even} \\ b_{(i-1)/2} & \text{if } i \text{ is odd} \end{cases}$$

**Q:** If we are being fussy, there is a small problem with this argument. What is it?

- **Proposition:** If there is a function  $f : \mathcal{N} \rightarrow S$  that is onto  $S$  then  $S$  is countable.

# Countable Unions of Countable Sets

- **Proposition:** The union of countably many countably infinite sets is countably infinite



$(0, 0),$   
 $(0, 1), (1, 0),$   
 $(0, 2), (1, 1), (2, 0),$   
 $\dots$

Each element is “reached” eventually in this ordering

**Q:** What is the bijection  $\mathcal{N} \leftrightarrow \mathcal{N} \times \mathcal{N}$ ?

# Are there uncountable sets? (Infinite but not countably infinite)

**Theorem:**  $\mathcal{P}(\mathcal{N})$  is uncountable  
(The set of all sets of natural numbers)

**Proof by contradiction:** (i.e. assume that  $\mathcal{P}(\mathcal{N})$  is countable and show that this results in a contradiction)

- ▶ Suppose that  $\mathcal{P}(\mathcal{N})$  were countable.
- ▶ There there is an enumeration of all subsets of  $\mathcal{N}$  say  $\mathcal{P}(\mathcal{N}) = \{S_0, S_1, \dots\}$

# Diagonalization

	$j = 0$	1	2	3	4	$\dots$
$S_i$						
$S_0$	Y	N	N	Y	N	$\dots$
$S_1$	N	N	N	N	N	$\dots$
$S_2$	Y	Y	N	Y	Y	$\dots$
$S_3$	N	N	N	Y	N	$\dots$
$\vdots$						

$D$

“Y” in row  $i$ , column  $j$  means  $j \in S_i$

- ▶ Let  $D = \{i \in \mathcal{N} : i \in S_i\}$  be the diagonal
- ▶  $D = YNNY\dots = \{0, 3, \dots\}$
- ▶ Let  $\overline{D} = \mathcal{N} - D$  be its complement
- ▶  $\overline{D} = NYYN\dots = \{1, 2, \dots\}$
- ▶ **Claim:**  $\overline{D}$  is omitted from the enumeration, contradicting the assumption that every set of natural numbers is one of the  $S_i$ s.
- ▶ **Pf:**  $\overline{D}$  is different from each row; they differ at the diagonal.

# Cardinality of Languages

- ▶ An alphabet  $\Sigma$  is finite by definition
- ▶ **Proposition:**  $\Sigma^*$  is countably infinite
- ▶ So every language is either finite or countably infinite
- ▶  $\mathcal{P}(\Sigma^*)$  is uncountable, being the set of subsets of a countable infinite set.

i.e. There are uncountably many languages over any alphabet

**Q:** Even if  $|\Sigma| = 1$ ?

# Existence of Non-regular Languages

**Theorem:** For every alphabet  $\Sigma$ , there exists a non-regular language over  $\Sigma$ .

**Proof:**

- ▶ There are only countably many regular expressions over  $\Sigma$ .  
     $\Rightarrow$  There are only countably many regular languages over  $\Sigma$ .
- ▶ There are uncountably many languages over  $\Sigma$ .
- ▶ Thus at least one language must be non-regular.

$\Rightarrow$  In fact, “almost all” languages must be non-regular.

**Q:** Could we do this proof using DFAs instead?

**Q:** Can we get our hands on an *explicit* non-regular language?



# Non-Regular Languages

**Reading:** Sipser, §1.4.

# Goal: Explicit Non-Regular Languages

It *appears* that a language such as

$$\begin{aligned} L &= \{x \in \Sigma^* : |x| = 2^n \text{ for some } n \geq 0\} \\ &= \{a, b, aa, ab, ba, bb, aaaa, \dots, bbbb, aaaaaaaa, \dots\} \end{aligned}$$

can't be regular because the “gaps” in the set of possible lengths become arbitrarily large, and no DFA could keep track of them.

But this isn't a proof!

## Approach:

1. Prove some general property  $P$  of all regular languages.
2. Show that  $L$  does **not** have  $P$ .

# Pumping Lemma (Basic Version)

If  $L$  is regular, then there is a number  $p$  (the **pumping length**) such that every string  $s \in L$  of length at least  $p$  can be divided into  $s = xyz$ , where  $y \neq \varepsilon$  and for every  $n \geq 0$ ,  $xy^n z \in L$ .

 $n = 1$ 

$x$	$y$	$z$
-----	-----	-----

 $n = 0$ 

$x$	$z$
-----	-----

 $n = 2$ 

$x$	$y$	$y$	$z$
-----	-----	-----	-----

 $\dots$

# Pumping Lemma (Basic Version)

If  $L$  is regular, then there is a number  $p$  (the **pumping length**) such that every string  $s \in L$  of length at least  $p$  can be divided into  $s = xyz$ , where  $y \neq \varepsilon$  and for every  $n \geq 0$ ,  $xy^n z \in L$ .

 $n = 1$ 

$x$	$y$	$z$
-----	-----	-----

 $n = 0$ 

$x$	$z$
-----	-----

 $n = 2$ 

$x$	$y$	$y$	$z$
-----	-----	-----	-----

...

- ▶ Why is the part about  $p$  needed?
- ▶ Why is the part about  $y \neq \varepsilon$  needed?

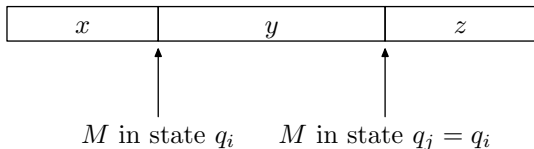
# Proof of Pumping Lemma

(Another fooling argument)

- ▶ Since  $L$  is regular, there is a DFA  $M$  accepting  $L$ .
- ▶ Let  $p = \#$  states in  $M$ .
- ▶ Suppose  $s \in L$  has length  $l \geq p$ .
- ▶  $M$  passed through a sequence of  $l + 1 > p$  states while accepting  $s$  (including the first and last states): say,  $q_0, \dots, q_l$ .
- ▶ Two of these states must be the same: say,  $q_i = q_j$  where  $i < j$

## Pumping, continued

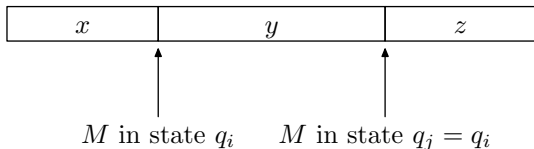
- ▶ Thus, we can break  $s$  into  $x, y, z$  where  $y \neq \varepsilon$  (though  $x, z$  may equal  $\varepsilon$ ):



- ▶ If more copies of  $y$  are inserted,  $M$  “can’t tell the difference,” i.e., the state entering  $y$  is the same as the state leaving it.
- ▶ So since  $xyz \in L$ , then  $xy^n z \in L$  for all  $n$ .

## Pumping, continued

- ▶ Thus, we can break  $s$  into  $x, y, z$  where  $y \neq \varepsilon$  (though  $x, z$  may equal  $\varepsilon$ ):



- ▶ If more copies of  $y$  are inserted,  $M$  “can’t tell the difference,” i.e., the state entering  $y$  is the same as the state leaving it.
- ▶ So since  $xyz \in L$ , then  $xy^n z \in L$  for all  $n$ .

### Proof also shows:

- ▶ We can take  $p = \#$  states in smallest DFA recognizing  $L$ .
- ▶ Can guarantee division  $s = xyz$  satisfies  $|xy| \leq p$  (or  $|yz| \leq p$ ).

# Pumping Lemma Example

- ▶ Consider

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

- ▶ Since  $L$  is regular, pumping lemma holds.  
(i.e., every sufficiently long string  $s$  in  $L$  is “pumpable”)
- ▶ For example, if  $s = aab$ , we can write  $x = \varepsilon$ ,  $y = aa$ , and  $z = b$ .



# Pumping the even $a$ 's, odd $b$ 's language

**Claim:**  $L$  satisfies pumping lemma with pumping length  $p = 4$ .

**Proof:**

## Pumping the even $a$ 's, odd $b$ 's language

**Claim:**  $L$  satisfies pumping lemma with pumping length  $p = 4$ .

### Proof:

Consider any string  $s$  of length at least 4, and write  $s = tu$  where  $|t| = 4$

- ▶ Case 1:  $t$  has an even number of  $a$ 's and an even number of  $b$ 's. Then we can set  $x = \varepsilon$ ,  $y = t$ ,  $z = u$ .
- ▶ Case 2:  $t$  has 3  $a$ 's and 1  $b$ . Then we can set  $y = aa$ .
- ▶ Case 3:  $t$  has 3  $b$ 's and 1  $a$ . Then we can set  $y = bb$ .
- ▶ So  $L$  satisfies the pumping lemma with pumping length  $p = 4$ .

**Q:** Can the Pumping Lemma be used to prove that  $L$  is regular?  
That is, does “Pumpable”  $\Rightarrow$  Regular?

# Use PL to Show Languages are *NOT* Regular

**Claim:**  $L = \{a^n b^n : n \geq 0\} = \{\varepsilon, ab, aabb, aaabbb, \dots\}$  is not regular.

## Proof by contradiction:

- ▶ Suppose that  $L$  is regular.
- ▶ So  $L$  has some pumping length  $p > 0$ .
- ▶ Consider the string  $s = a^p b^p$ . Since  $|s| = 2p > p$ , we can write  $s = xyz$  for some strings  $x, y, z$  as specified by the lemma.
- ▶ Claim: No matter how  $s$  is partitioned into  $xyz$  with  $y \neq \varepsilon$ , we have  $xy^2z \notin L$ .
- ▶ This violates the conclusion of the pumping lemma, so our assumption that  $L$  is regular must have been false.

# Strings of exponential lengths are a nonregular language

**Claim:**  $L = \{w : |w| = 2^n \text{ for some } n \geq 0\}$  is not regular.

**Proof:**

# Strings of exponential lengths are a nonregular language

**Claim:**  $L = \{w : |w| = 2^n \text{ for some } n \geq 0\}$  is not regular.

## Proof:

- ▶ Suppose  $L$  satisfies the pumping lemma with pumping length  $p$ .
- ▶ Choose any string  $s \in L$  of length greater than  $p$ , say  $|s| = 2^n$ .  
By pumping lemma, write  $s = xyz$ .
- ▶ Let  $|y| = k$ . Then  $2^n - k, 2^n, 2^n + k, 2^n + 2 \cdot k, \dots$  are all powers of two.
- ▶ This is impossible. QED.

# “Regular Languages Can’t Do Unbounded Counting”

**Claim:**  $L = \{w : w \text{ has the same number of } a\text{'s and } b\text{'s}\}$  is not regular.

**Proof #1:**

- ▶ Use pumping lemma on  $s = a^p b^p$  with  $|xy| \leq p$  condition.

# “Regular Languages Can’t Do Unbounded Counting”

**Claim:**  $L = \{w : w \text{ has the same number of } a\text{'s and } b\text{'s}\}$  is not regular.

## Proof #1:

- ▶ Use pumping lemma on  $s = a^p b^p$  with  $|xy| \leq p$  condition.

## Proof #2:

- ▶ If  $L$  were regular, then  $L \cap a^* b^*$  would also be regular.

# Reprise on Regular Languages

Which of the following are necessarily regular?

- ▶ A finite language
- ▶ A union of a finite number of regular languages
- ▶  $\{x : x \in L_1 \text{ and } x \notin L_2\}$ ,  $L_1$  and  $L_2$  are both regular
- ▶ A subset of a regular language



# What Happens During the Transformations?

- ▶ NFA  $\rightarrow$  DFA
- ▶ DFA  $\rightarrow$  Regular Expression
- ▶ Regular Expression  $\rightarrow$  NFA

# Minimizing DFAs

Many different DFAs accept the same language. But there is a smallest one—and we can find it!

- ▶ Let  $M$  be a DFA
- ▶ Say that states  $p, q$  of  $M$  are **distinguishable** if there is a string  $w$  such that exactly one of  $\delta^*(p, w)$  and  $\delta^*(q, w)$  is final.
- ▶ Start by dividing the states of  $M$  into two equivalence classes: the final and non-final states.

# Minimizing DFAs, continued

- ▶ Break up the equivalence classes according to this rule: If  $p, q$  are in the same equivalence class but  $\delta(p, \sigma)$  and  $\delta(q, \sigma)$  are not equivalent for some  $\sigma \in \Sigma$ , then  $p$  and  $q$  must be separated into different equivalence classes.
- ▶ When all the states that must be separated have been found, form a new and finer equivalence relation.
- ▶ Repeat.
- ▶ How do we know that this process stops?