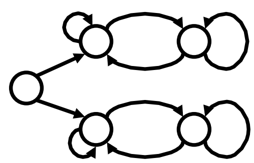
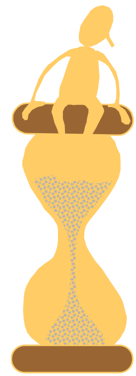
Great Theoretical Ideas In Computer Science		
Anupam Gupta Danny Sleator		CS 15-251 Fall 2010
Lecture 20	Oct 28, 2010	Carnegie Mellon University

Finite Automata

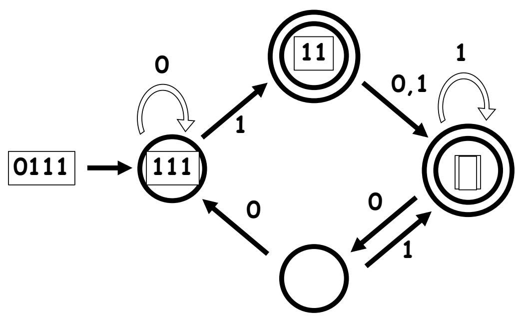


Deterministic Finite Automata

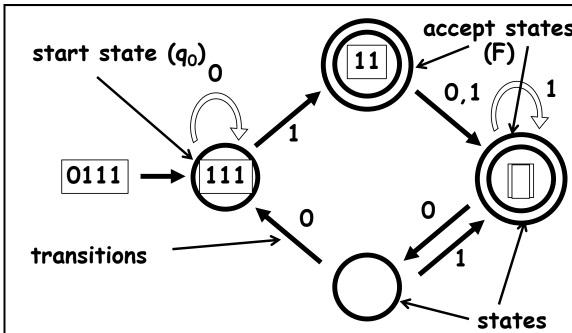


A machine so simple that you can understand it in less than one minute

Wishful thinking…



The machine accepts a string if the process ends in a double circle



The machine accepts a string if the process ends in a double circle

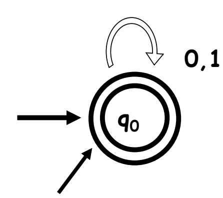
Anatomy of a Deterministic Finite Automaton

The singular of automata is automaton.

The alphabet of a finite automaton is the set where the symbols come from, for example {0,1}

The language of a finite automaton is the set of strings that it accepts

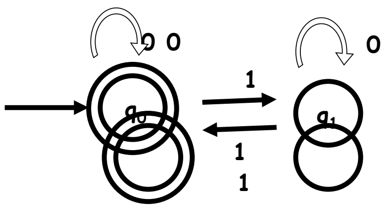
The Language L(M) of Machine M



L(M) = All strings of 0s and 1s

The language of a finite automaton is the set of strings that it accepts

The Language L(M) of Machine M



L(M) = { w | w has an even number of 1s}

Notation

An alphabet Σ is a finite set (e.g., Σ = {0,1})

A string over Σ is a finite-length sequence of elements of Σ

For x a string, |x| is the length of x

The unique string of length 0 will be denoted by ε and will be called the empty or null string

A language over Σ is a set of strings over Σ

A finite automaton is $M = (Q, \Sigma, \delta, q_0, F)$

- Q is the finite set of states
- Σ is the alphabet
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of accept states

L(M) = the language of machine M
= set of all strings machine M accepts

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

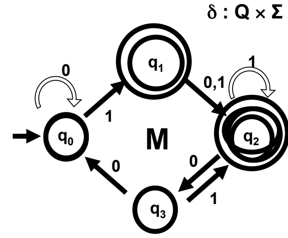
$Q = \{q_0, q_1, q_2, q_3\}$

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

$q_0 \in Q$ is start state

$F = \{q_1, q_2\} \subseteq Q$ accept states

$\delta : Q \times \Sigma \rightarrow Q$ transition function

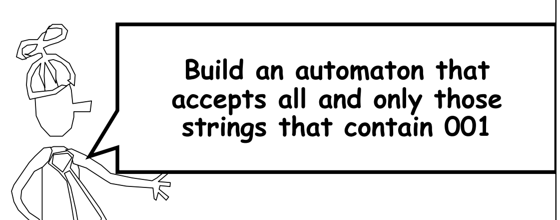


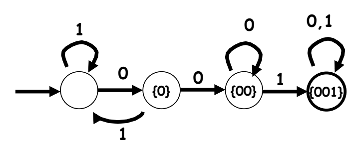
δ	0	1
q_0	q_0	q_1
q_1	q_2	q_2
q_2	q_3	q_2
q_3	q_0	q_2

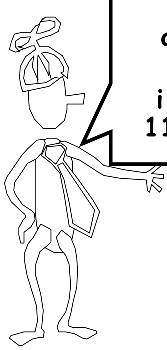
The finite-state automata are deterministic, if for each pair $Q \times \Sigma$ of state and input value there is a unique next state given by the transition function.

There is another type machine in which there may be several possible next states. Such machines called nondeterministic.

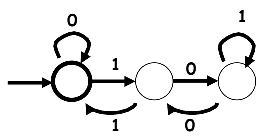
EXAMPLE







Build an automaton that accepts all binary numbers that are divisible by 3, i.e, L = 0, 11, 110, 1001, 1100, 1111, 10010, 10101…

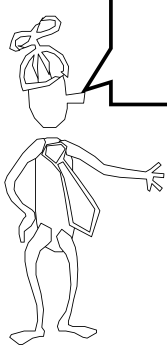


A language over Σ is a set of strings over Σ

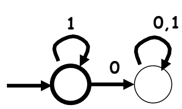
A language is regular if it is recognized by a deterministic finite automaton

L = { w | w contains 001} is regular

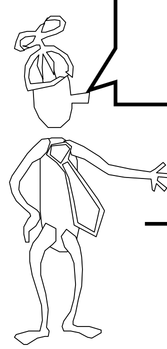
L = { w | w has an even number of 0s} is regular



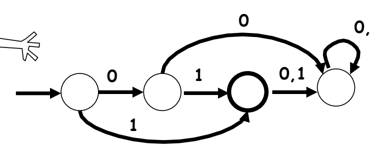
Determine the language recognized by



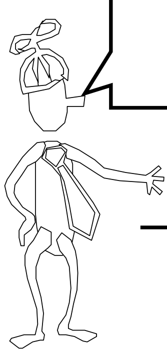
L(M)={1^n | n = 0, 1, 2, …}



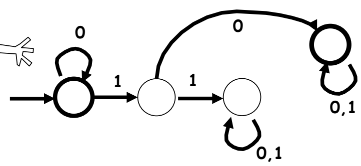
Determine the language recognized by



L(M)={1, 01}



Determine the language recognized by



L(M)={0^n, 0^n10x | n=0,1,2…, and x is any string}

DFA Membership problem

Determine whether some word belongs to the language.

Theorem: The DFA Membership Problem is solvable in linear time.

Let M = (Q, Σ, δ, q_0, F) and w = $w_1 \ldots w_m$.
Algorithm for DFA M:

p := q_0;
for i := 1 to m do p := δ(p,w_i);
if p∈F then return Yes else return No.

Equivalence of two DFAs

Definition: Two DFAs M_1 and M_2 over the same alphabet are equivalent if they accept the same language: $L(M_1) = L(M_2)$.

Given a few equivalent machines, we are naturally interested in the smallest one with the least number of states.

Union Theorem

Given two languages, L_1 and L_2, define the union of L_1 and L_2 as

$L_1 \cup L_2 = \{ w \mid w \in L_1 \text{ or } w \in L_2 \}$

Theorem: The union of two regular languages is also a regular language.

Theorem: The union of two regular languages is also a regular language

Proof (Sketch): Let

$M_1 = (Q_1, \Sigma, \delta_1, q_0^1, F_1)$ be finite automaton for L_1

and

$M_2 = (Q_2, \Sigma, \delta_2, q_0^2, F_2)$ be finite automaton for L_2

We want to construct a finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ that recognizes $L = L_1 \cup L_2$

Idea: Run both M_1 and M_2 at the same time

Q = pairs of states, one from M_1 and one from M_2

= $\{ (q_1, q_2) \mid q_1 \in Q_1 \text{ and } q_2 \in Q_2 \}$

= $Q_1 \times Q_2$

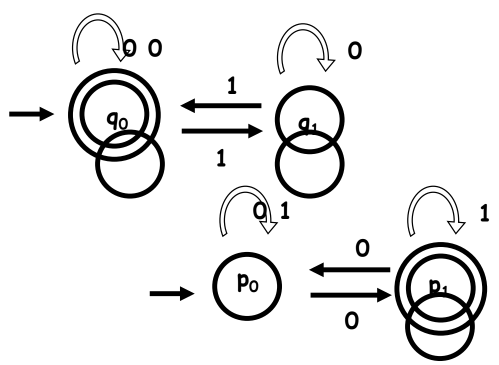
$q_0 = (q_0^1, q_0^2)$

$\delta((q_1,q_2),\sigma) = (\delta_1(q_1,\sigma), \delta_2(q_2,\sigma))$

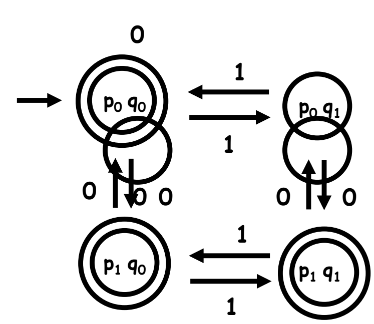
$F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$

Easy to see that this simulates both machines and accepts the union. QED

Theorem: The union of two regular languages is also a regular language



Automaton for Union



The Regular Operations

Union: $A \cup B = \{ w \mid w \in A \text{ or } w \in B \}$

Intersection: $A \cap B = \{ w \mid w \in A \text{ and } w \in B \}$

Negation: $\neg A = \{ w \mid w \notin A \}$

Reverse: $A^R = \{ \sigma_1 \ldots \sigma_k \mid \sigma_k \ldots \sigma_1 \in A \}$

Concatenation: $A \cdot B = \{ vw \mid v \in A \text{ and } w \in B \}$

Star: $A^* = \{ w_1 \ldots w_k \mid k \geq 0 \text{ and each } w_i \in A \}$

Reverse

Reverse: $A^R = \{ \sigma_1 \ldots \sigma_k \mid \sigma_k \ldots \sigma_1 \in A \}$

How to construct a DFA for the reversal of a language?

The direction in which we read a string should be irrelevant. If we flip transitions around we might not get a DFA.

The Kleene closure: A*

Star: $A^* = \{ w_1 \ldots w_k \mid k \geq 0 \text{ and each } w_i \in A \}$

From the definition of the concatenation, we define A^n, n =0, 1, 2, … recursively

$A^0 = \{\varepsilon\}$

$A^{n+1} = A^n A$

A* is a set consisting of concatenations of any number of strings from A.

$$A^* = \bigcup_{1 \leq k < \infty} A^k$$

The Kleene closure: A*

What is A* of A={0,1}?

All binary strings

What is A* of A={11}?

All binary strings of an even number of 1s

Regular Languages Are Closed Under The Regular Operations

We have seen the proof for Union. You will prove some of these on your homework.

Theorem: Any finite language is regular

Claim 1: Let w be a string over an alphabet. Then {w} is a regular language.

Proof: Construct the automaton that accepts {w}.

Claim 2: A language consisting of n strings is regular

Proof: By induction on the number of strings. If {a} then $L \cup \{a\}$ is regular

Pattern Matching

Input: Text T of length t, string S of length n

Problem: Does string S appear inside text T?

Naïve method:

$\boxed{a_1, a_2, a_3,} a_4, a_5, \ldots, a_t$

Cost: Roughly nt comparisons

Automata Solution

Build a machine M that accepts any string with S as a consecutive substring

Feed the text to M

Cost: t comparisons + time to build M

As luck would have it, the Knuth, Morris, Pratt algorithm builds M quickly

Real-life Uses of DFAs

Regular Expressions

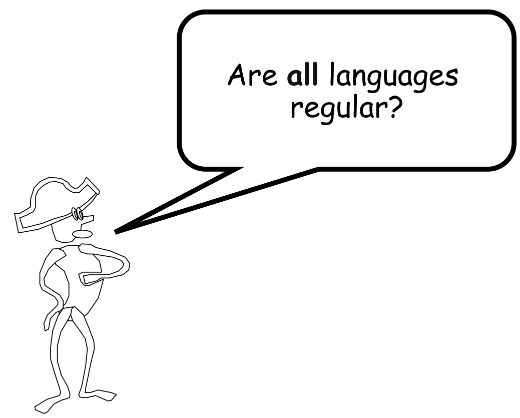
Coke Machines

Thermostats (fridge)

Elevators

Train Track Switches

Lexical Analyzers for Parsers

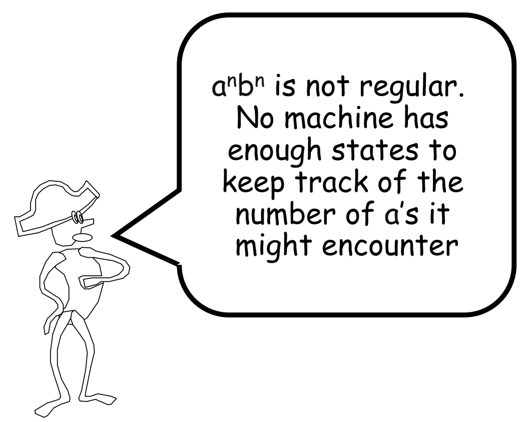


Consider the language L = { a^nb^n | n > 0 }

i.e., a bunch of a's followed by an equal number of b's

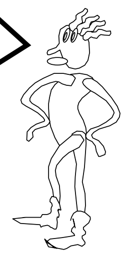
No finite automaton accepts this language

Can you prove this?



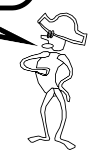
That is a fairly weak argument

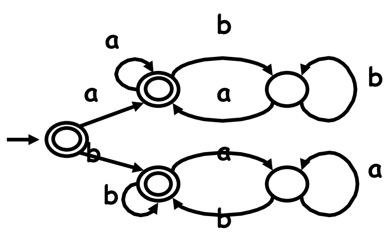
Consider the following example…



L = strings where the # of occurrences of the pattern ab is equal to the number of occurrences of the pattern ba

Can't be regular. No machine has enough states to keep track of the number of occurrences of ab

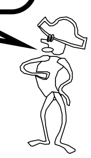




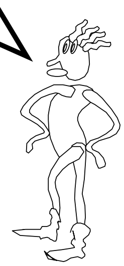
M accepts only the strings with an equal number of ab's and ba's!

L = strings where the # of occurrences of the pattern ab is equal to the number of occurrences of the pattern ba

~~Can't be regular. No machine has enough states to keep track of the number of occurrences of ab~~



Let me show you a professional strength proof that $a^n b^n$ is not regular…



How to prove a language is not regular…

Assume it is regular, hence is accepted by a DFA M with n states.

Show that there are two strings s_1 and s_2 which both reach some state in M (usually by pigeonhole principle)

Then show there is some string t such that string s_1t is in the language, but s_2t is not. However, M accepts either both or neither.

Theorem: L= $\{a^n b^n \mid n > 0\}$ is not regular

Proof (by contradiction):

Assume that L is regular, $M=(Q,\{a,b\},\delta,q_0,F)$

Consider $\delta(q_0, a^i)$ for i = 1,2,3, …

There are infinitely many i's but a finite number of states.

$\delta(q_0, a^n)=q$ and $\delta(q_0, a^m)=q$, and $n \neq m$

Since M accepts $a^n b^n$ $\delta(q, b^n)=q_f$

$\delta(q_0, a^m b^n)=\delta(\delta(q_0, a^m),b^n)=\delta(q, b^n)= q_f$

It follows that M accepts $a^m b^n$, and $n \neq m$

The finite-state automata are deterministic, if for each pair of state and input value there is a unique next state given by the transition function.

There is another type machine in which there may be several possible next states. Such machines called nondeterministic.

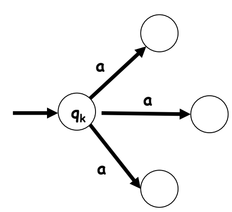
Nondeterministic finite automaton (NFA)

A NFA is defined using the same notations
$M = (Q, \Sigma, \delta, q_0, F)$
as DFA except the transition function δ assigns a set of states to each pair $Q \times \Sigma$ of state and input.

A string is accepted iff there exists some set of choices that leads to an accepting state

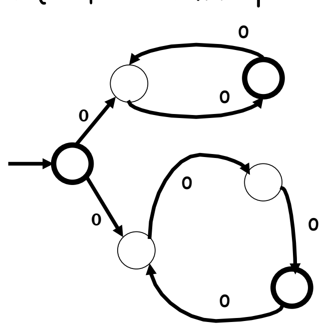
Note, every DFA is automatically also a NFA.

Nondeterministic finite automaton

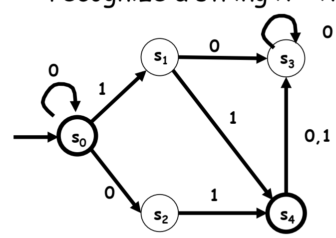


Allows transitions from q_k on the same symbol to many states

NFA for $\{0^k \mid k \text{ is a multiple of 2 or 3}\}$



What does it mean that for a NFA to recognize a string $x = x_1x_2\ldots x_k$?



Since each input symbol x_j (for j>1) takes the previous state to a set of states, we shall use a union of these states.

What does it mean that for a NFA to recognize a string?

Here we are going formally define this.

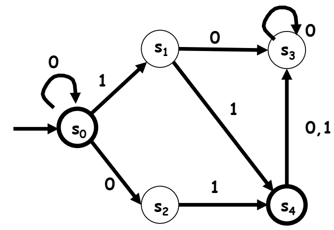
For a state q and string w, $\delta^*(q, w)$ is the set of states that the NFA can reach when it reads the string w starting at the state q.

Thus for NFA= $(Q, \Sigma, \delta, q_0, F)$, the function

$$\delta^*: Q \times \Sigma^* \to 2^Q$$

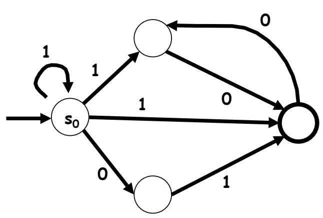
is defined by $\delta^*(q, y\, x_k) = \bigcup_{p \in \delta^*(q,y)} \delta(p, x_k)$

Find the language recognized by this NFA



L = $\{0^n, 0^n01, 0^n11 \mid n = 0, 1, 2\ldots\}$

Find the language recognized by this NFA



L = 1^* (01, 1, 10) $(00)^*$

Theorem: The languages accepted by an NFA are regular.

In other words:
For any NFA there is an equivalent DFA.

This theorem may prove useful on the homework. You should prove it if you want to use it.

NFA vs. DFA

NFA
Richer notation to represent a language.
Sometimes exponentially smaller.

DFA
Implementable in low level hardware.
Very fast to simulate.

Study Bee

- DFAs
- Regular Languages
- Regular operators
- a^nb^n is not regular
- NFAs
- NFAs accept regular languages