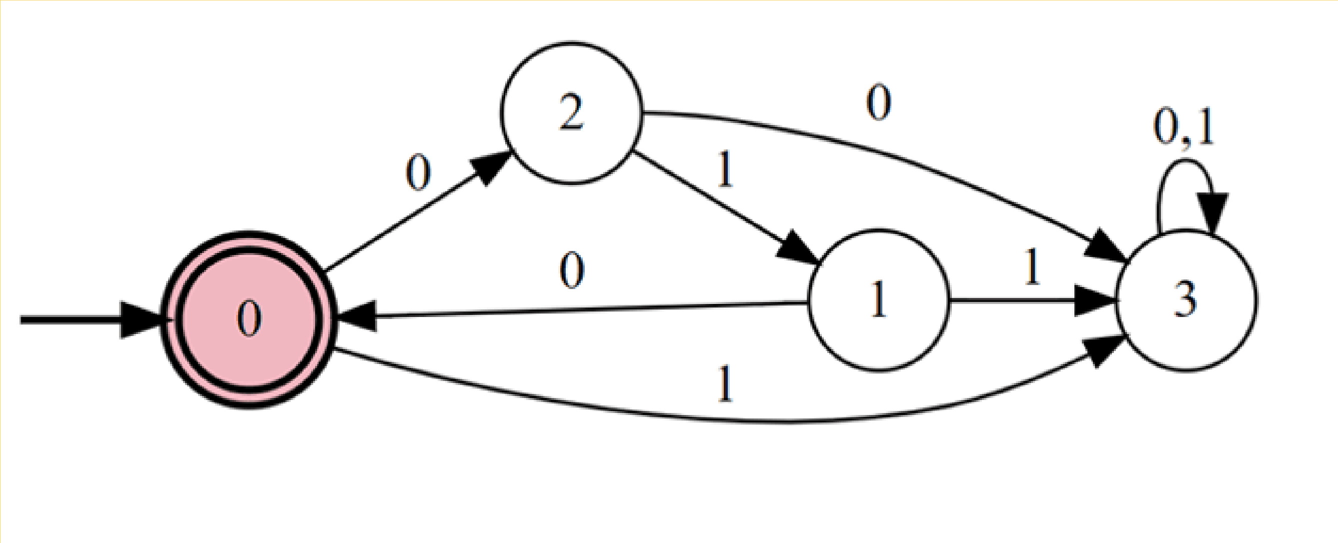
Problem Set 5:
Will be posted this week

~~posted~~ today
due next Fri



Class 12:
Finite Automata, Regular Expression

University of Virginia
cs3120: DMT2
Wei-Kai Lin

Recap: Functions with Infinite Domains

That is, $\{0,1\}^*$

$f : \{0,1\}^* \rightarrow \{0,1\}$ with **one bit** of output is called a **Boolean** function.

$\{0,1\}^* = \{\epsilon, 0,1,00,01,10,11, \dots\}$ is an **infinite set**, but it **does not** contain any **infinitely long input**.

Terminology: “Language”

Definition (Language).
A Boolean function $f : \{0,1\}^* \rightarrow \{0,1\}$ defines a corresponding set $L_f = \{x \mid f(x) = 1\} \subseteq \{0,1\}^*$, also called a **language**.

So, $f(x) = 1 \quad \leftrightarrow \quad$ “x belongs to language L_f”

Deterministic Finite Automata

A simple computing model

Fixed constant-size memory

Reading input $x = x_1 \dots x_n$ as a stream of bits (once)

Decide if $f(x) = 1$ or not at the end

The same “automata” works for all input length n

What's Automata?

automaton noun

au·tom·a·ton ȯ-ˈtä-mə-tən -mə-ˌtän

plural **automatons** *or* **automata** ȯ-ˈtä-mə-tə -mə-ˌtä

1 : a mechanism that is relatively self-operating

especially : **ROBOT**

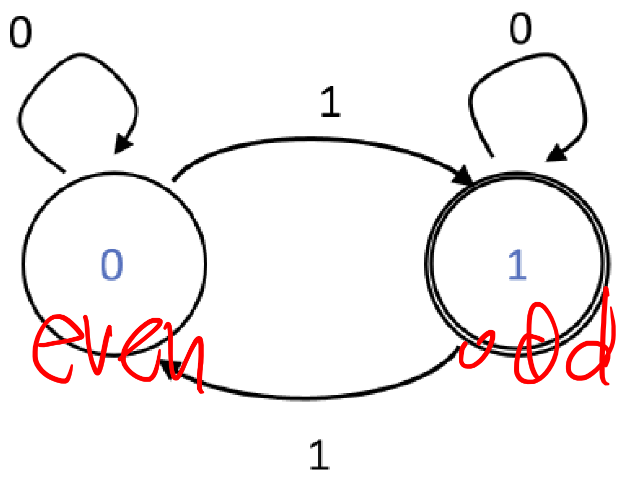
2 : a machine or control mechanism designed to follow automatically a predetermined sequence of operations or respond to encoded instructions

Machine

Example: XOR

For $x = x_1 \ldots x_n$ let $XOR(x) = x_1 \oplus x_2 \ldots \oplus x_n$

As we read through the bits, we only need one bit of memory b that determines the XOR so far.



Formal Definition of FA

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***.

Introduction to the Theory of Computation. Sipser 2006.

Formal definition of FAs accepting inputs

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. $r_n \in F$.

Constant-size memory: only needs r_i
Efficient: read each w_i only once

Example:

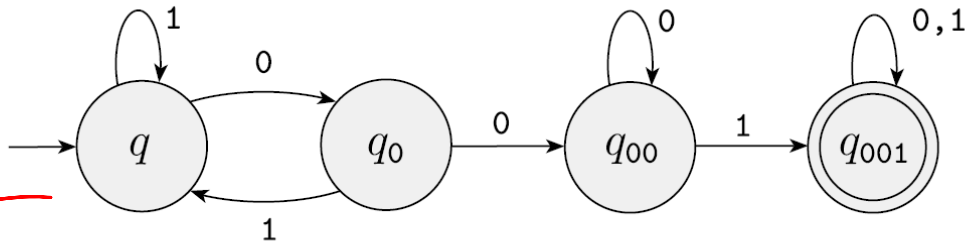


FIGURE 1.22
Accepts strings containing 001

1. $Q = \boxed{q, q_0, q_{00}, q_{001}}$
2. $\Sigma = \{0,1\}$,
3. δ is described as

state	input 0	1
q	q_0	q
q_0	q_{00}	q
q_{00}	q_{00}	q_{001}
q_{001}	q_{001}	q_{001}

accept

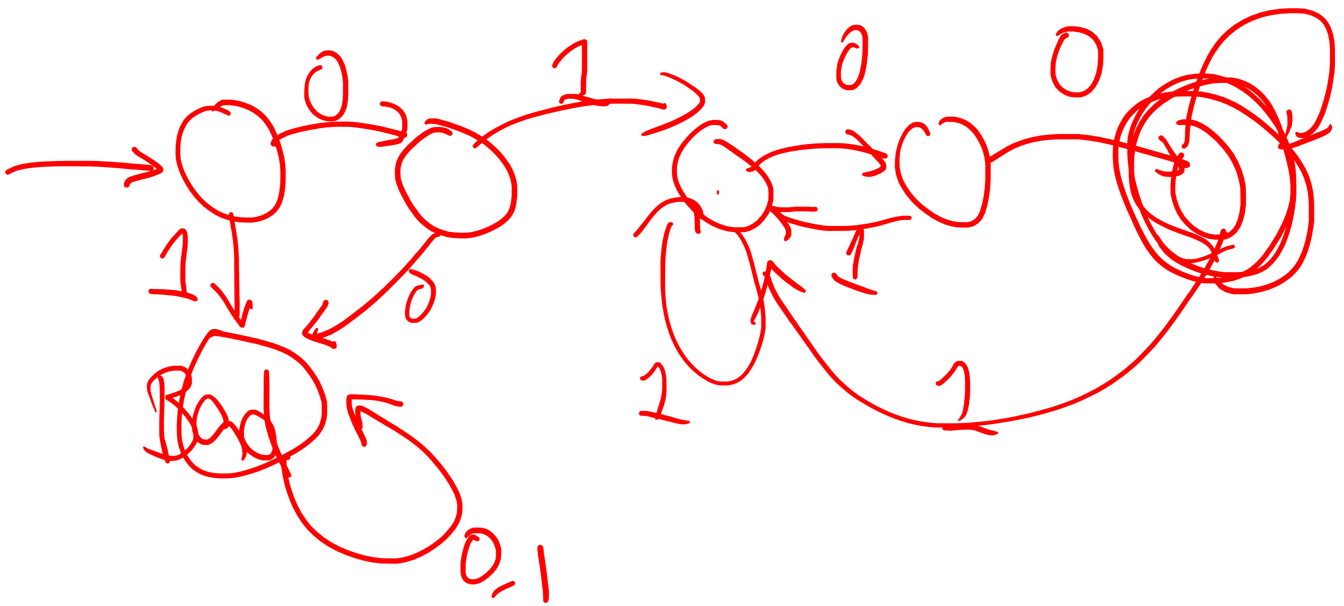
4. $\boxed{q}$ is the start state, and
5. $F = \boxed{q_{001}}$

Example: prefix and suffix

Alphabet $\Sigma = \{0,1\}$. All strings with:

Prefix: 01 Suffix: 00

01 00



01001

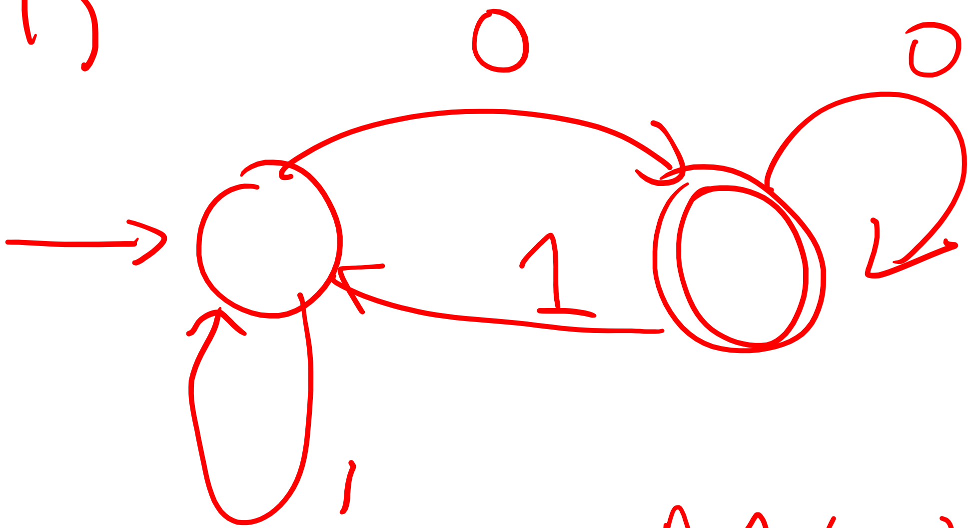
010010

Another example: set of even numbers

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

even nat ~~num~~

suf "0"



$EVEN = \{a \mid a \text{ is even}\}$

$$M(x) = \begin{cases} 1 & \text{if } x \in EVEN \\ 0 & \text{o.w.} \end{cases}$$

x a sequence of int/nat

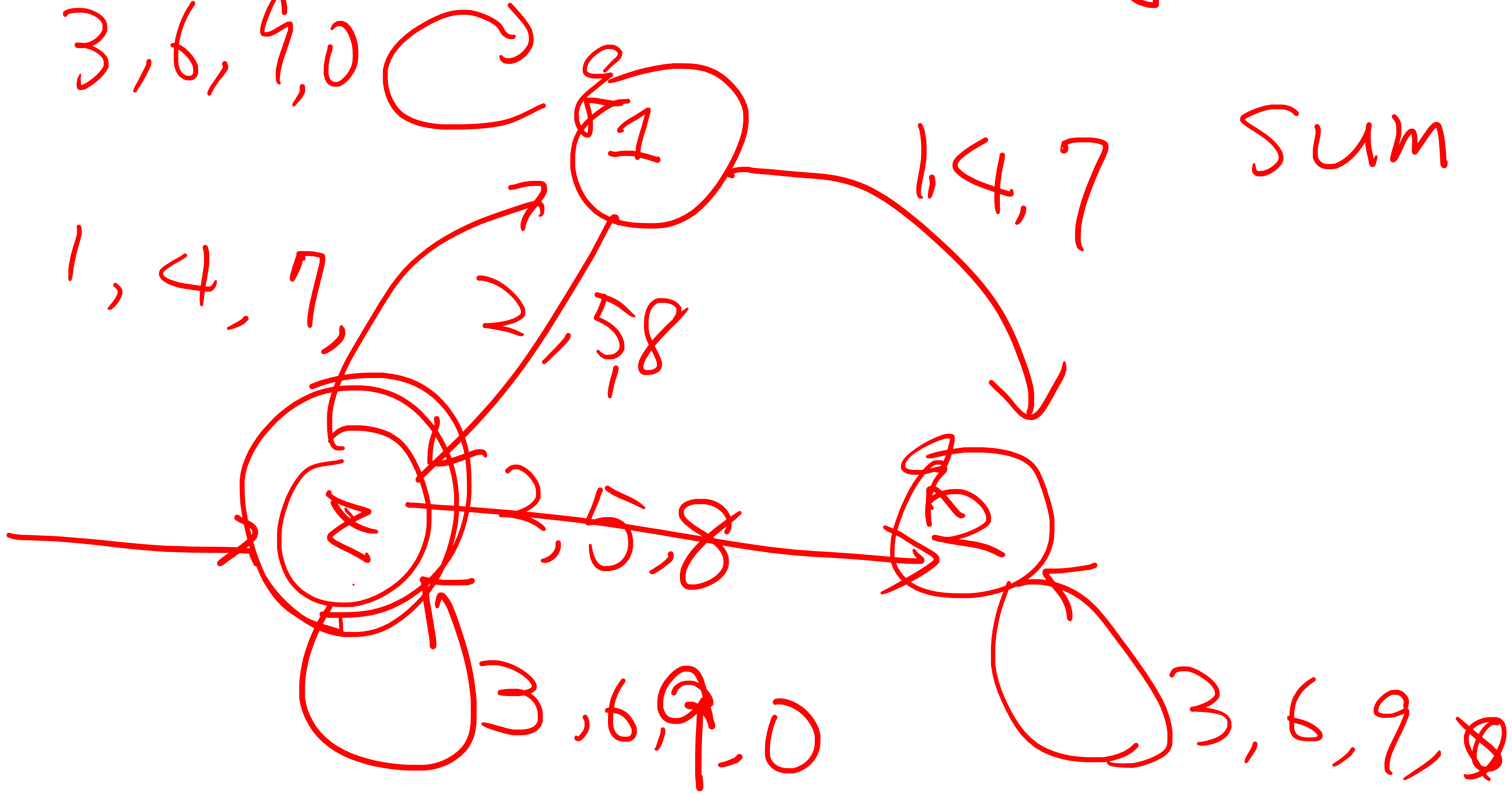
$= (x_1, x_2 \cdots x_t)$

$M(x)$ s.t. x_i aren all $i = 1 \ldots t$

Example: Set of multiples of 3 in basis 10:

$\Sigma = \{0, \dots, 9\}$ 0901 = 901

Lemma: Sum of digits is multiple 3.



sum $d_1 + \dots d_i$ mod 3

3 states.

Recap the objects + terminology

Let M be a FA
Let f be the function: $f(x) = 1$ iff M accepts x
We say that M "computes" or "decides" f

Let L be the language $x \in L$ iff $f(x) = 1$
We might say M "recognizes" (or "decides") the language L (or function f)

f, L ignore the computation's details, but M cares about it

Complexity class: DFA-Comp

DFA-Comp $= \{f \mid f \text{ is computed by some DFA } M\}$

More generally "Complexity class" for **set of algorithms $\boldsymbol{X}$**:
= all languages/functions computable with an algorithm in the set X.

DFA-Comp = Complexity class of "FAs"

Recall $SIZE_n$ = complexity class for functions computable by size-s circuits.

Regular expressions

A (seemingly) completely different way of dealing with infinite languages
(i.e., functions on infinite input sets)

Motivation and example

Suppose we want to describe key-words for search

Simple examples: “book” or “February” or “a” or “10”
Intense example: a sorted sequence of integers

Maybe crazy: a very large integer that is prime

Can we support all above? Python prog
Do we want to support all?

Motivation: simple rules

How about patterns with simple rules.

Example: we want a string of zeros only or ones only of length at least 2

Kleene Star

We denote them as: $(00(0^*)|11(1^*))$

- 00 simply means string 00
- 0^* means repeating 0 zero, or one, or two, or … times.
- | means OR

Regular Expression is Widely Used

Examples:

*.pdf: any string ends with ‘.pdf’

?.pdf: any string ends with ‘.pdf’ that is exactly 5 chars

Wildcards is a subset of regular expressions.

Writing “regular expressions” using “regular operations”

Definition 6.6 (Regular expression)

A *regular expression* e over an alphabet Σ is a string over $\Sigma \cup \{(,),|,*,\emptyset,""\}$ that has one of the following forms:

1. $e = \sigma$ where $\sigma \in \Sigma$ $= \{0, 1\}$
2. $e = (e'|e'')$ where e', e'' are regular expressions. OR
3. $e = (e')(e'')$ where e', e'' are regular expressions. (We often drop the parentheses when there is no danger of confusion and so write this as $e'\ e''$.) Concatenate
4. $e = (e')^*$ where e' is a regular expression. rep

Finally we also allow the following “edge cases”: $e = \emptyset$ and $e = ""$. These are the regular expressions corresponding to accepting no strings, and accepting only the empty string respectively.

Example: prefix and suffix

Alphabet $\Sigma = \{0,1\}$. All strings with:

Prefix: 01 Suffix: 00

$\cancel{2}\,01\,00$

01001

$r = (01)(0|1)^*(00)$

$(0|1)^*(0)^*$

01 00 ?

01 ? 00

Example: XOR

For $x = x_1 \ldots x_n$ let $XOR(x) = x_1 \bigoplus x_2 \ldots \bigoplus x_n$

Observation: there are **odd** number of ‘1’s

Parenthesis and Precedence

- Drop parentheses when inferred from context

- highest precedence to
  $*$, then
  concatenation, and then
  OR

- $00^{*} \mid 11$ instead of $((0)(0^{*})) \mid ((1)(1))$

Regular Expressions as Functions

For every regular expression e,
there is a corresponding function $\Phi_e : \{0,1\}^* \rightarrow \{0,1\}$

Such that

$$\Phi_e(x) = 1 \text{ if } \; x \text{ matches } e.$$

Defining Φ_e defines the evaluation of e.

Recursive definition, but cumbersome.

Definition 6.6 (Regular expression)

A *regular expression* e over an alphabet Σ is a string over $\Sigma \cup \{(,),|,*,\emptyset,""\}$ that has one of the following forms:

0. $e = \emptyset$ or $e =$ ""
1. $e = \sigma \in \Sigma$

2. $e = (e'|e'')$

3. $e = (e')(e'')$

4. $e = (e')^*$

e', e'' are regular expressions

Definition 6.7 (Matching a regular expression)

Let e be a regular expression over the alphabet Σ. The function $\Phi_e : \Sigma^* \to \{0,1\}$ is defined as follows:

1. If $e = \sigma$ then $\Phi_e(x) = 1$ iff $x = \sigma$.
2. If $e = (e'|e'')$ then $\Phi_e(x) = \Phi_{e'}(x) \vee \Phi_{e''}(x)$ where $\vee$ is the OR operator.
3. If $e = (e')(e'')$ then $\Phi_e(x) = 1$ iff there is ~~some~~ $x', x'' \in \Sigma^*$ such that x is the concatenation of x' and x'' and $\Phi_{e'}(x') = \Phi_{e''}(x'') = 1$. any
4. If $e = (e')*$ then $\Phi_e(x) = 1$ iff there is some $k \in \mathbb{N}$ and some $x_0, \ldots, x_{k-1} \in \Sigma^*$ such that x is the concatenation $x_0 \cdots x_{k-1}$ and $\Phi_{e'}(x_i) = 1$ for every $i \in [k]$.
5. Finally, for the edge cases $\Phi_\emptyset$ is the constant zero function, and $\Phi_{""}$ is the function that only outputs 1 on the empty string "".

We say that a regular expression e over Σ *matches* a string $x \in \Sigma^*$ if $\Phi_e(x) = 1$.

An (Python) algorithm evaluates regular expression

Charge

Deterministic Finite Automata

Regular Expressions

PRR 6: will be release later today, due next Monday, 10pm
PS5: due next Friday, Feb 28, 10pm