Finite State Machines

Chang-Gun Lee (cglee@snu.ac.kr)

Assistant Professor

The School of Computer Science and Engineering

Seoul National University

Information Processing Machine

- A machine that takes input data, processes them, and produces output data
- Examples
  - Table lamp

up,	down,	down,	up,	down, …
light,	dark,	dark,	light,	dark, …

  - Adder

3,	5,	0,	3,	3, …
4,	4,	6,	1,	4, …
7,	9,	6,	4,	7, …

  - Vending machine

dime,	dime,	dime,	quarter,	quarter,	nickel,	quarter,	nickel, …
nothing,	nothing,	gum,	nothing,	gum,	nothing,	gum,	nothing

  - Digital Computer

Difference between Adder and VM?

- Adder: The output signal at any instant depends only on the input signal at that instant.
- Vending machine: The output signal at any instant depends not only on the input signal at that instant but also on the preceding input signals
  - the vending machine is capable of remembering the total amount that has been deposited.

Total deposit	New deposit			output
	nickel	dime	quarter	
0	5	10	25	nothing
5	10	15	30 or more	nothing
10	15	20	30 or more	nothing
15	20	25	30 or more	nothing
20	25	30 or more	30 or more	nothing
25	30 or more	30 or more	30 or more	nothing
30 or more	5	10	25	gum

Two classes of machines

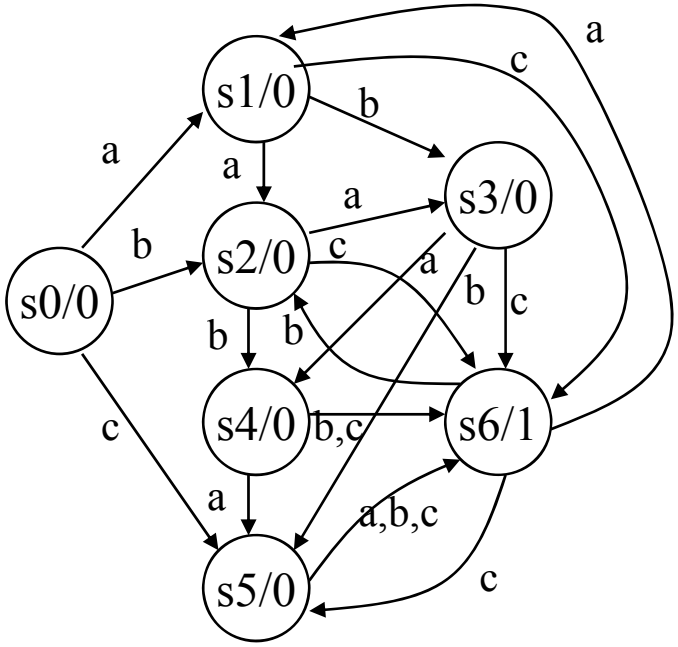
- Class A: machines without memory
- Class B: machines with memory
- Even for a machine with memory, it does not remember everything that has happened in the past. It remembers only a summary (or abstraction) of the past history
- We call such a summary a “state”.
- How to define states of the vending machine?

- A machine with a finite number of states is called a *finite state machine*.

Finite State Machines

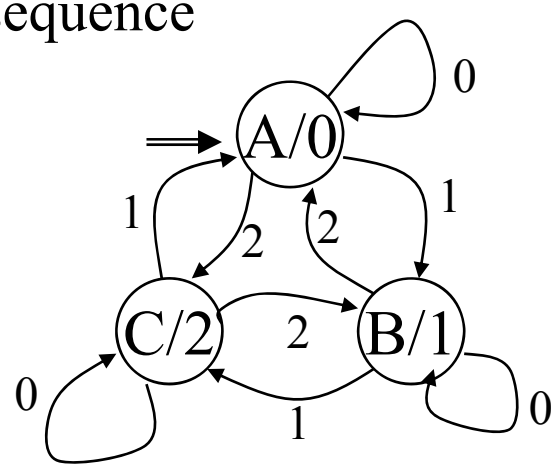
- A finite state machine is specified by
  - A finite set of states $S = \{s_0, s_1, s_2, \ldots\}$
  - A special element of the set S, s_0, referred to as the *initial state*.
  - A finite set of input letters $I=\{i_1,i_2, \ldots\}$.
  - A finite set of output letters $O=\{o_1, o_2, \ldots\}$
  - A function f from $S \times I$ to S, referred to as the *transition function*.
  - A function g from S to O, referred to as the *output function*.

State	input			output
	a	b	c	
s0	s1	s2	s5	0
s1	s2	s3	s6	0
s2	s3	s4	s6	0
s3	s4	s5	s6	0
s4	s5	s6	s6	0
s5	s6	s6	s6	0
s6	s1	s2	s5	1



FSM as Models of Physical Systems

- Modulo 3 counter: a FSM that receives a sequence of 0s, 1s, and 2s as input and produces a sequence of 0s, 1s, and 2s as output such that at any instant, the output is equal to the modulo 3 sum of the digits in the input sequence



- Comparator: a FSM that receives two binary numbers and determine whether they are equal, lager, or smaller. We assume that the digits of the two numbers come in one by one, with the lower-order digits coming in first.

State					output
	00	01	10	11	
A	A	C	B	A	Equal
B	B	C	B	B	Larger
C	C	C	B	C	Smaller

Equivalent Machines

- Two FSMs are said to be equivalent if, starting from their respective initial states, they will produce the same output sequence when they are given the same input sequence.

state	input		output
	1	2	
A	B	C	0
B	F	D	0
C	G	E	0
D	H	B	0
E	B	F	1
F	D	H	0
G	E	B	0
H	B	C	1

state	input		output
	1	2	
A	B	C	0
B	C	D	0
C	D	E	0
D	E	B	0
E	B	C	1

C=F, D=G, E=H

- Two states s_i and s_j are said to be equivalent if for any input sequence the machine will produce the same output sequence whether it starts in s_i or s_j.

How do we know equivalent states?

- Two states are said to be *0-equivalent* if they have the same output.
- Two states are said to be *1-equivalent* if they have the same output and if, for ever input letter, their successor are 0-equivalent.
- Two states are said to be *k-equivalent* if they have the same output and if, for every input letter, their successors are (k-1)-equivalent.
- Two states are equivalent if they are k-equivalent for all k.

state	input		output
	0	1	
A	B	F	0
B	A	F	0
C	G	A	0
D	H	B	0
E	A	G	0
F	H	C	1
G	A	D	1
H	A	C	1

k-equivalence makes a equivalent relation among states!

$\pi_0=\{\overline{ABCDE}\ \ \overline{FGH}\}$

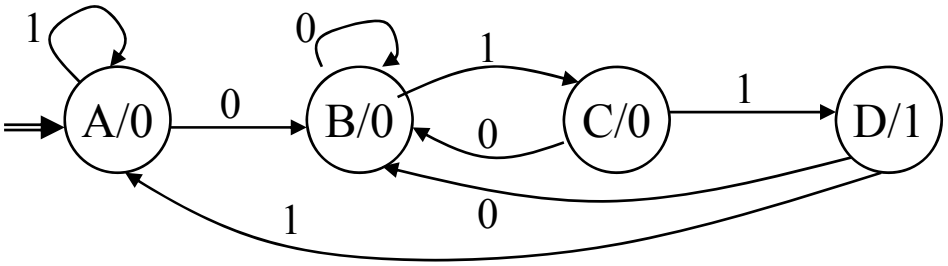
$\pi_1=\{\overline{ABE}\ \ \overline{CD}\ \ \overline{F}\ \ \overline{GH}\}$

$\pi_2=\{\overline{AB}\ \ \overline{CD}\ \ \overline{E}\ \ \overline{F}\ \ \overline{GH}\}$

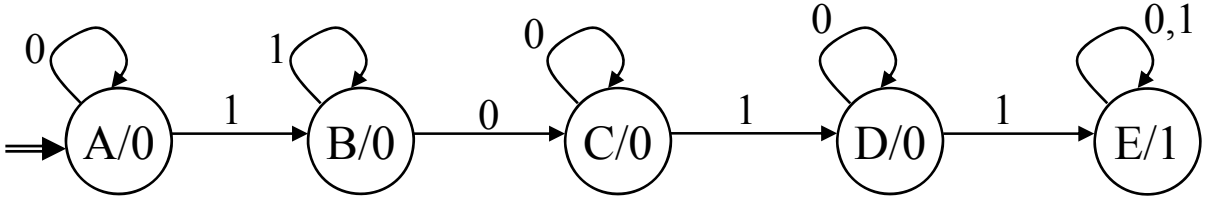
$\pi_3=\{\overline{AB}\ \ \overline{CD}\ \ \overline{E}\ \ \overline{F}\ \ \overline{GH}\}$

FSMs as Language Recognizers

- A FSM with
  - accepting states (output = 1)
  - rejecting states (output = 0)
- An input sequence is said to be accepted by the FSM if it leads the machine from the initial state to an accepting state, and said to be rejected, otherwise.
- Example 1: Make a FSM that accepts all binary sequences that ends with the digits 011.

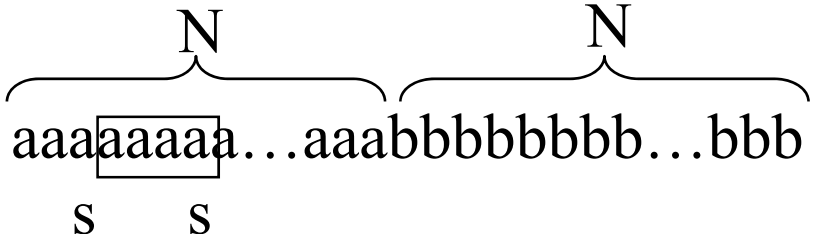


- Example 2: Make a FSM that accepts all binary sequences of the form any number of 0s, followed by one or more 1s, followed by one or more 0s, followed by a 1, followed by any number of 0s, followed by a 1, and then followed by anything.



Finite State Language

- A language is said to be a finite state language (or a regular language) if there is a finite state machine that accepts exactly all sequences in the language
- Any FSM defines a finite state language.
- A given language might or might not be a finite state language.
- Consider the language $L=\{a^k b^k | k \geq 1\}$. Is it a finite state language?
  - No
  - Prove by contradiction
    - Pumping lemma



Finite State Languages and Type-3 Languages

- Surprisingly, a finite state language is a type-3 language and a type-3 language is a finite state language.
- To show this, let's introduce a nondeterministic FSM.
  - A finite set of states $S = \{s_0, s_1, s_2, \ldots\}$
  - A special element of the set S, s_0, referred to as the *initial state*.
  - A finite set of input letters $I=\{i_1,i_2, \ldots\}$.
  - A finite set of output letters $O=\{o_1, o_2, \ldots\}$
  - A function f from $S \times I$ to 2^S, referred to as the *transition function*.
  - A function g from S to O, referred to as the *output function*.

state	input		output
	0	1	
A	B	B,C	0
B	A,C	C	0
C	A	B,C	1

	0	0	0	1	0	0	0	0	1
A	B	A	A	B	A	A	A	A	B
		C	B	C	C	B	B	B	C
							C	C	

Nondeterministic FSM as Language Recognizer

- We say that a sequence is accepted by a nondeterministic FSM if starting from the initial state, among all the final states the sequence will lead the machine into, one of them is an accepting state.
- Is a nondeterministic FSM is more powerful than a deterministic FSM in the sense that there are languages that can be recognized by a nondeterministic machine but cannot be recognized by a deterministic one?
  - No, For any given nondeterministic FSM, there is a deterministic finite state machine that accepts exactly the same language.

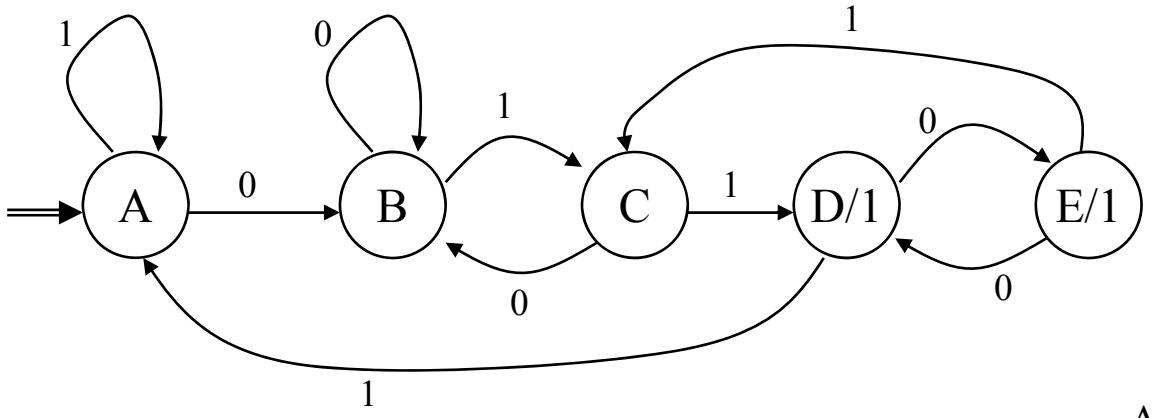
state	input		output
	0	1	
A	B	B,C	0
B	A,C	-	0
C	A	B,C	1

state	input		output
	0	1	
{A}	{B}	{B,C}	0
{B}	{A,C}	{ }	0
{C}	{A}	{B,C}	1
{A,B}	{A,B,C}	{B,C}	0
{A,C}	{A,B}	{B,C}	1
{B,C}	{A,C}	{B,C}	1
{A,B,C}	{A,B,C}	{B,C}	1
{ }	{ }	{ }	0

Nondeterministic FSMs and Type-3 Languages

- The class of finite state languages is exactly the class of type-3 languages.
- We show
  - Given a FSM, we can have a type-3 grammar specifying the language accepted by the FSM.
  - Given a type-3 grammar, we can construct a nondeterministic FSM that accepts the language specified the grammar.

Constructing Type-3 Grammar from FSM



A→0B
A→1A
B→0B
B→1C
C→0B
C→1D
C→1
D→0E
D→0
D→1A
E→0D
E→0
E→1C

Constructing FSM from Type-3 Gramamr

A→0A
A→1B
B→0C
B→0D
C→0
C→1B
C→1D
D→1
D→1A