Recursive Definitions

Chapter - II

Chapter - No-1

* Definition and types of grammars and Languages

*1) Recursive definitions of Functions with Domain N

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n * (n-1)! & \text{if } n > 0 \end{cases}$$

Here we define the factorial of any term at $n=0$ then by using $n=0$ find $n=1, 2, \ldots n$

eg for Mathematical Induction

$$0+1+2+\cdots n = \frac{n(n+1)}{2}$$

Now take

1) $n=0$

$$= \frac{0(0+1)}{2}$$

2) $n=1 \quad = 0+1 \quad = \frac{0(0+1)}{2} + 1$

by using result of $n=0$

ie $\frac{0(0+1)}{2}$

Now. eg.

1) Fibonacci Function

The fibonacci funn of following

$f(0) = 1$

$f(1) = 1$

$f(2) = f(1) + f(0) = 2$

Two Methods

1) Top down
2) Bottom Up

① Calculate $f(4)$ by top down & bottom up.

1) Top Down

$f(4) = f(3) + f(2)$

$= f(2) + f(1) + f(1) + f(0)$

$= f(1) + f(0) + f(1) + f(1) + f(0)$

$= 1 + 1 + 1 + 1 + 1$

$\therefore \boxed{f(4) = 5}$

By Top Down

Find. $f(7)$

Bottom - Up fashion

$f(0) = 1$
$f(1) = 1$
$f(2) = f(1) + f(0) = 2$
$f(3) = f(2) + f(1) = 2+1 = 3$
$f(4) = f(3) + f(2) = 3+2 = 5$

By fabonacci funin f is usually defined as

$f(0) = 1$
$f(1) = 1$

$$\text{for every } n \geq 1 \quad f(n+1) = f(n) + f(n-1)$$

step by induction.

Now

Poove that for every $n \geq 0$

$$f(n) \leq (5/3)^n$$

① Basis step :-

Now at $n=0$

$f(0) = 1$

at $f(n) = (5/3)^n$

$(5/3)^0 = 1$

$\therefore$ $f(0) = (5/3)^0$

for $k \geq 0$ with n

$0 \leq n \leq k$.

$\therefore \boxed{f(n) \leq (5/3)^n}$

Statement to show in induction step.

$f(k+1) \leq (5/3)^{k+1}$

Proof of Induction step :-

Now for every $k \geq 0$

$f(k+1) = f(1) = 1$

Now $f(k+1) = f(k) + f(k-1)$

$= \left(\frac{5}{3}\right)^k + \left(\frac{5}{3}\right)^{k-1}$

$= (5/3)^{k-1}\left[\frac{5}{3} + 1\right]$

$= \left(\frac{5}{3}\right)^{k-1}\left(\frac{8}{3}\right)$

$= \left(\frac{5}{3}\right)^{k-1}\left(\frac{24}{9}\right)$

$$f(k+1) = \left(\frac{5}{3}\right)^{k-1}\left(\frac{24}{9}\right)$$

$$< \left(\frac{5}{3}\right)^{k-1}\left(\frac{25}{9}\right)$$

$$< \left(\frac{5}{3}\right)^{k-1}\left(\frac{5}{3}\right)^{2}$$

$$f(k+1) < \left(\frac{5}{3}\right)^{k+1}$$

$$\therefore \boxed{f(k+1) \leq \left(\frac{5}{3}\right)^{k+1}}$$

Recursive Definitions of sets

It can't involve an integer explicitly n expli principle is similar. We specify certain objects that can set to start with & describe one or more general method.

** Definition & types of grammers & Languages

Language :- set of string involves symbols from alphebet. & number

string should be

-> set of alphebets from a to z & A to Z

Alphebets are set of finite 26 symbols

Numbers -> 0 to 9 Numbers.

3) A string over alphebet Σ is obtained by placing some elements of Σ

if x is string then over Σ & it mention no of symbols in it ie $|x|$ mod x.

string over alphebet {a, b} are a, ab, aa, aba etc.

we have

$|a| = 1$

$|ab| = |aa| = 2$

$|aba| = 3$

A null string denoted by Λ

- We never allow the string $\wedge$ ie null string

- for any Alphebet Σ the set of all string over Σ is denoted by Σ^*

if $\Sigma = \{a, b\}$ then

$$\Sigma^* = \{a, b\}^*$$

$$= \{\wedge, a, b, aa, ab, ba, bb, aaa, aab, aba, abb \ldots\}$$

- For Any two string over alphebet Σ the following operations could be perform
  1) Union
  2) intersection
  3) Difference

Universal set - Σ^*

If L is language then subset

$$L' = \Sigma^* - L$$

$$L_1 \subseteq \Sigma_1^*$$

$$L_2 \subseteq \Sigma_2^*$$

then L_1 & L_2 are both subset of $(\Sigma_1 \cup \Sigma_2)^*$

Concatenation

if x & y are elements of Σ^* concatenation could be xy

concatenation is purely associative

if $x = aab$

$y = bba$

$xy = aabbba$

$yx = bbaaab$

- Concanetation operation should be words on two strings like

if $L_1 . L_2 \subseteq \Sigma^*$

$$L_1 L_2 = \{xy \mid x \in L_1 \text{ and } y \in L_2\}$$

eg

{ hope, fear } { less, fully } =

{ hopeless, hopefully, fearless, fearfully }

- Concatenation of a string x with null $\wedge$ will give x

eg.

$a^0 = \wedge$

$x^0 = \wedge$

$\Sigma^0 = \{\wedge\}$

$L^0 = \{\wedge\}$

which are analogous to algebra in algebra

$a^0 = 1$ & so on.

Now Unit of Concantenation is $\wedge$ ie

if $x^p x^q = x^{p+q}$

if $p = 0$ then

$x^0 x^q = \wedge \cdot x^q = x^q$

ie $\wedge$ is unit of multiplication

- Let us consides concatenating the sting is times then it will be L^k.

- If infinite no of operations are performed then

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

$* \Rightarrow$ kleene. star.

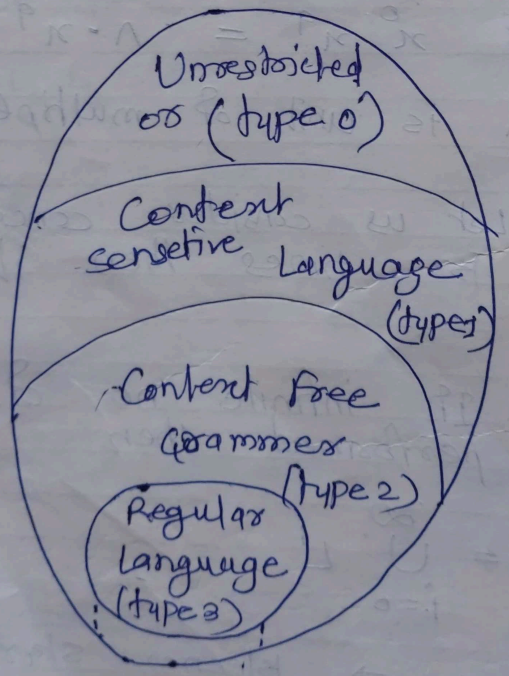
The given string starts from 0 & having infinite symbols

if the string start with 1 then it will simply represented by

$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

Here L^+ & L^+ are same

* ⁴ Chomskey Relation in Language & Grammer



Types of Language

1) Context sensentive language (type1)
2) Context free Grammer (type 2)
3) Regular language type (type3)
4) Unrestricted or type 0

① Unrestricted or type 0

Unrestricted Grammer is 4 tuples

$G = \{V, \Sigma, S, P\}$

V = finite set of Variables

Σ = Terminal symbol

$S \in \Sigma$ start symbol

P = finite set of Productions

Unrestricted Grammer permits productions of the form

$$\alpha \rightarrow \beta$$

α = string of terminal & Non Terminal with at least one non terminal & not ϵ

β = string of terminal & Nonterminal

eg. $AE \rightarrow \epsilon$

$AB \rightarrow CD$

2) Context - sensitive Grammer / Langu
- Contain four tuple else type 0 & called as Content sensitive or type 1 grammer.

if $\alpha \rightarrow \beta$

α = sting of terminal & Non terminal (at least one)

β = At least as long as α

$bB \rightarrow bb$

$AE \rightarrow AB$

$aA \rightarrow aa$

3) Content free Language

- Every Content free is content sensetive
- Reverse of content free is content free
- It perform Union Concatenation & Kleene Operation

4) Regular Language :- It Contain all the properties of type 0, 1 & 2 lang

* Regular Expressions & Corresponding Regular Languages

Regular Expression -

The algebric notation describes exactly the same language as finite automata.

It contain operations like

1) Union
2) Concatenation (dot)
3) closure (star)

Algebric Operations

	union	Concatenation	closure
Associative	✓	✓	×
commutative	✓	×	×
idempotent.	✓	×	×

While Converting Regular language into regular expression following rules are followed

1) Replace set brackets {} by parantheses 2) , by U.
3) U by +

The result will be regular expression

Regular Language	Corresponding Regular Expression
1) $\{\wedge\}$	$(\wedge) = \wedge$
2) $\{0\}$	0
3) $\{001\}$ ie $\{0\}\{0\}\{1\}$	001
4) $\{0, 10\}$ ie $\{0\} \cup \{10\}$	$0+10$
5) $\{1\}^* \{10\}$	1^*10
6) $\{10, 111, 11010\}^*$	$(10+111+11010)^*$
7) $\{110\}^* \{0, 1\}$	$(110)^*(0+1)$

Defn **Regular Languages & Regular Expression over Σ**

The set R of Regular languages over Σ & the corresponding regular expression are defined as follows

1) If ϕ is element of R. then corresponding regular expression is ϕ

2) $\{\wedge\}$ is element of R the corresponding regular expression is $\wedge$.

3) for each $a \in \Sigma$ $\{a\}$ is an element of R then corresponding regular expression is a

4) If L_1 & L_2 are elements of R & corresponding regular expression is $(r_1 + r_2)$

a) $L_1 \cup L_2$ is an element of R., corresponding Regular expression is $(r_1 + r_2)$;

b) $L_1 L_2$ is an element of R., and the corresponding regular expression is $(r_1 r_2)$;

c) L_1^* is an element of R. and the corresponding regular expression is (r_1^*)

r^2 stands for (rr)

(r^+) stands for $((r^*)r)$

* kleene having highest precedence
+ having lowest
concanetuling in between them

$(a+b)^* \neq a^* + b^*$

Rules for simplifing regular expression over $\{0,1\}$

$1^*(1+\wedge) = 1^*$

$1^*1^* = 1^*$

$0^* + 1^* = 1^* + 0^*$

$(0^*1^*)^* = (0+1)^*$

$(0+1)^*01(0+1)^* + 1^*0^* = (0+1)^*$

$(0+1)^*01 + 1^*0^*$

Examples in Regular Expression & Regular Language

1) string of even length

$\forall\ L \subseteq \{0,1\}^*$ be language of all string of even length

Any string of even length can be obtained by concatenating zero or more strings of length 2. having even length.

$L = \{00, 01, 10, 11\}^*$

Regular Expression corresponding to L is

$(00+01+10+11)^*$

$(0(0+1)+1(0+1))^*$

$((0+1)(0+1))^*$

2) : String of Length 6 or less.

Let L be the set of all string over $\{0,1\}$ of length 6 or less

Regular language

$L = \{\wedge, 0, 1, 00, 01, 11, 000, \ldots$

$\ldots 111110, 111111\}$

Converting into regular expression

$\wedge + 0 + 1 + 00 + 01 + \ldots + 111110 + 111111$

$\wedge + (0+1) + 0(0+1) + 1(0+1) + \ldots +$

$11111(0+1)$

$\wedge + (0+1)(0+1)(0+1)(0+1)(0+1)(0+1)$

By Neglecting All null ($\wedge$) Transitions we get

$(0+1)(0+1)(0+1)(0+1)(0+1)(0+$

$(0+1)^6$ Regular expression

by Considering it we get

$(\wedge+0+1)^6$

All or some term Contain negligible term equals to

$0+1 = \wedge+0+1$

3) Language of C Identifiers

It Contain

1) Digits (d)
2) letters (l)
3) underscores (_)

digits $d = 0+1+\cdots+9$

letters $l = a+b+\cdots+Y+Z$

Underscore (_)

$(l+\_)(l+d+\_)^*$

* Regular Expressions & Languages :-

- Concatenating of simple string having length 1, if language having $\{a\}$ then $a \in \Sigma$
- It contain only string of single character. Instead of concatenation we can add.

ϕ emply language
$\wedge$ null string.

- The regular language over alphebet Σ contain basic operations like
  union
  concatenation
  Kleene *
- we replace $\cup$ by + & $\{\}$ with parentheses the result called as regular expression.

Eg:-

	Regular Language	Regular Expression
1)	$\{\wedge\}$	$\wedge$
2)	$\{0\}$	0
3)	$\{001\}$ ie. $\{0\}\{0\}\{1\}$	001
4)	$\{0,1\}$ ie $\{0\} \cup \{1\}$	0+1

5) $\{0, 10\}$ ie $(\{0\} \cup \{10\})$ $0+10$

6) $\{1, \wedge\}\{001\} \Rightarrow$

$(1+\wedge)\,001$

7) $\{110\}^*\{0,1\} \Rightarrow (110)^*(0+1)$

8) $\{1\}^*\{10\} \Rightarrow 1^*10$

9) $\{10, 111, 11010\}^* \Rightarrow (10+111+11010)^*$

10) $\{0, 10\}^*(\{11\}^* \cup \{001, \wedge\}) \Rightarrow$

$(0+10)^*((11)^* + (001+\wedge))$

Regular Languages & Regular Expressions over Σ

The set R of Regular languages over Σ & corresponding regular expressions are.

1) ϕ is element of R. Then corresponding regular expression is ϕ

2) $\{\wedge\}$ is an elements of R the $\wedge$ is regular expression

3) For each $a \in \Sigma$, $\{a\}$ is an element of R. The regular expression will be a

4) If L_1 & L_2 are any elements of R. & r_1 & r_2 are corresponding regular expressions

1) $L_1 \cup L_2$ then regular expression $(r_1 + r_2)$

2) $L_1 L_2$ is element of R then regular expression $(r_1 r_2)$;

3) If L_1^* is element of R then corresponding regular expression will be (r_1^*)

Algebric Expressions

1) Kleene * having highest
2) concantenation having 2nd
3) + having last priority

we can write $(a + ((b^*) c))$ as

$a + b^* c$.

$(a+b)^* \neq a+b^*$

Poove $(r^* s^*)^* = (r+s)^*$

$1^*(1+\Lambda) = 1^*$

$1^* 1^* = 1^*$

$0^* + 1^* = 1^* + 0^*$

$(0^* 1^*)^* = (0+1)^*$

ie $(r^* s^*)^* = (r+s)^*$

Eg:- 1) Poove string of even length

$L = \{00, 01, 10, 11\}^*$

$= \{0(0+1)1(0+1)\}^*$

$\boxed{L = \{(0+1)(0+1)\}^*}$

2) The language of C identifiers

l = letters $a, b, \ldots z, A, \ldots$

d = digits $0, 1, \ldots 9$

$L = (a+b+\ldots z+A+B+\ldots Z+$

$0+1+\ldots+9)$

only letters, digits & underscores (-)

$(L + \_)(L + d + \_)^*$

⇒ strings of length 6 or less

Let L be string over $\{0,1\}$ having length 6 or less

$\therefore L = \Lambda + 0 + 1 + 00 + 01 + 10 + \ldots + 111111$

$= (0+1)(0+1)(0+1)(0+1)(0+1)(0+1)$

$= (0+1)^6$

it will written as

$(0+1+\Lambda)^6$

string other than 00.

* Applications of Regular Expression

1) Regular Expressions in UNIX –

Complex regular expressions & languages can be reduced by using RE.

eg { } replaced by ( )
, by U, U by + etc

$\{0,1\} \Rightarrow (0+1) \quad \forall \{0\} \cup \{1\} = (0+1)$

In above eg only two symbols so it is simple if it contain more symbols like 128 then it will be diff to write so we use Unix

Alphebet –

In language
$\{a, b, \ldots, z, A, B, \ldots, Z\}$

In Regular Expression
{ } replaced by ( ) & , by +

$(a+b+ \ldots +z+A+ \ldots +Z)$

In Unix RE
[a-z A-Z]

Numbers [0-9]

2) Lexical Analysis

- Oldest Application
- Scan source program & recognize all tokens
token (All keywords, identi
- Separation of source prog into tokens & give to destinatio called lexical analysing of progra

3) Finding Pattern in Text

High level by using very small effort

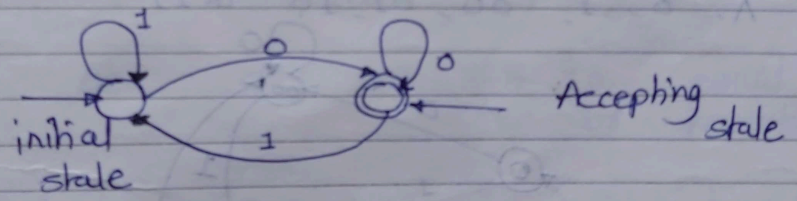
Compiles - Regular expression into executable code.

** Finite Automata

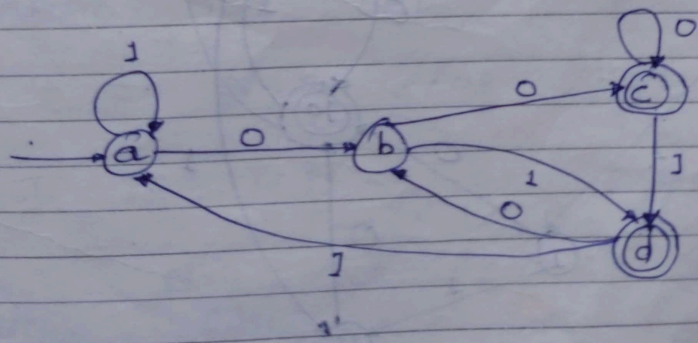
Finite Automata or Finite state machine (FA) is a 5 tuple (Q, Σ, q_0, A, δ) or (Q, Σ, q_0, F, δ)

Q ⇒ Finite set of states
Σ ⇒ Finite alphebet of i/p symbols
$q_0 \in Q$ → initial state
$A \subseteq Q$ → set of accepting states
δ is a fuin from $Q \times \Sigma$ to Q
$Q \times \Sigma \to Q$ ie transition function

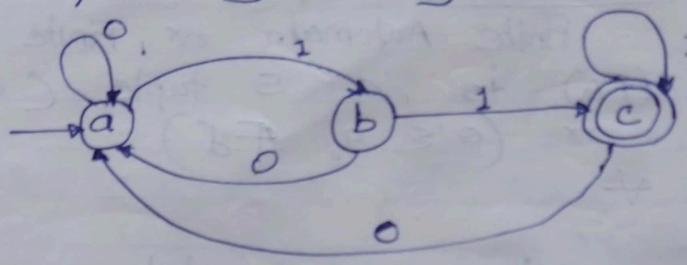
Eg:- ∀ string Ending in 0



⇒ string with next to last symbol 0



⇒ string Ending with 11



⇒ If $L = \{0,1\}^* \{10\}$ the langu
of all string in $\{0,1\}^*$ ending
with 10

→

Let we take the states l
Λ, 0, 1, 00, 01, 10 & 11

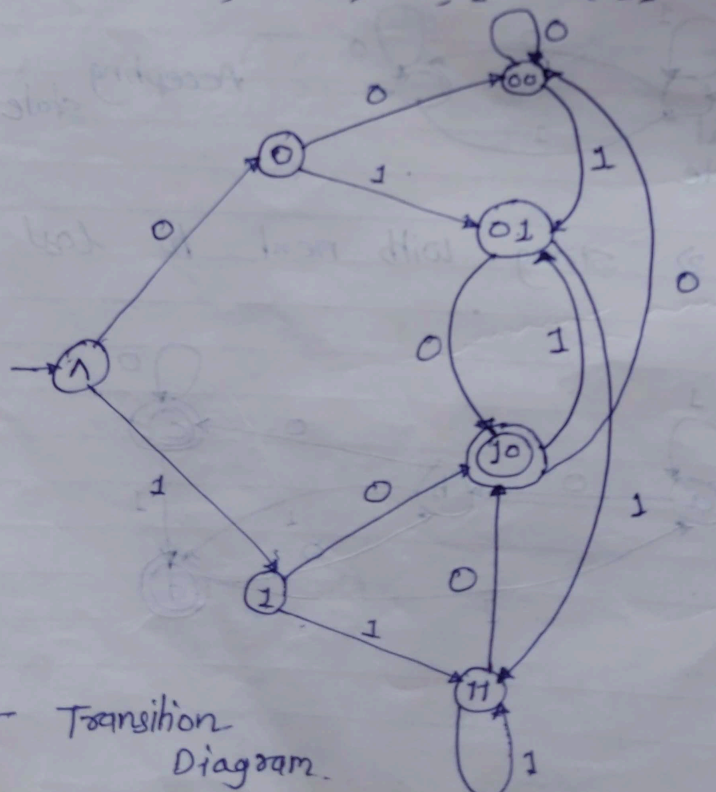


Fig:- Transition Diagram.

Transition Table

states	input 0	1
→ ^	0	1
0	00	01
1	10	11
00	00	01
01	10	11
(10)	00	01
11	10	11

identical states / transitions

Now from the given table we can conclude that

At 0, 00 & 10 equal i/p or transition

at 1, 01, & 11 equal transition

so

Above steps are identical from above we can mention

0, 00, & 10 as single step p.

1, 10 & 11 as single step q

Now we consider only three steps else

p, q & (10) Accepting state

Simplified finite Automaton recognizing $\{0,1\}^* \{10\}$ will be

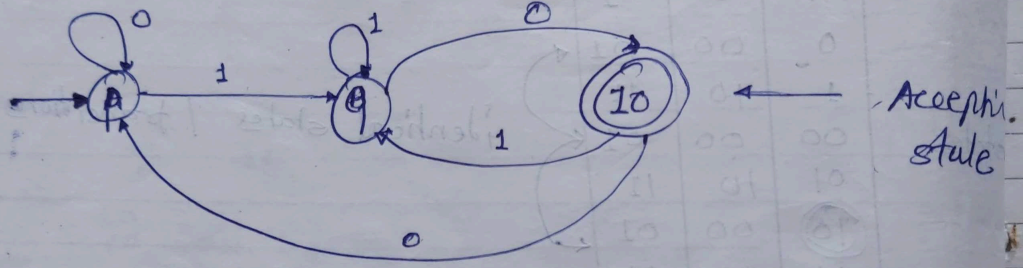


fig :- Transition Diagram

From Above The transition table will be

state	input 0	1
A	A	B
B	10	B
10	A	B

Fig :- Transition Table

If we consider null transitions then it will be

state	i/p 0	1
^	A	B
A	A	B
B	10	B
10	A	B

A = string end with 0

B = string end with 1

* Finite Automata

A finite automata is 5 tuple $(Q, \Sigma, q_0, A, \delta)$ called as finite state machine.

Q = finite set
Σ = finite element of i/p symbols
$q_0 \in Q$ initial state

$A \subseteq Q$ Accepting set
δ = transition function

Let M be the FA with Q state set

$\therefore M = (Q, \Sigma, q_0, A, \delta)$

Eg 1> $L = \{0, 1\}^* \{10\}$
strings ending with 10.

	0	1
$\wedge$	0	1
0	00	01
1	10	11
00	00	01
01	10	11
10	00	01
11	10	11

Transition table

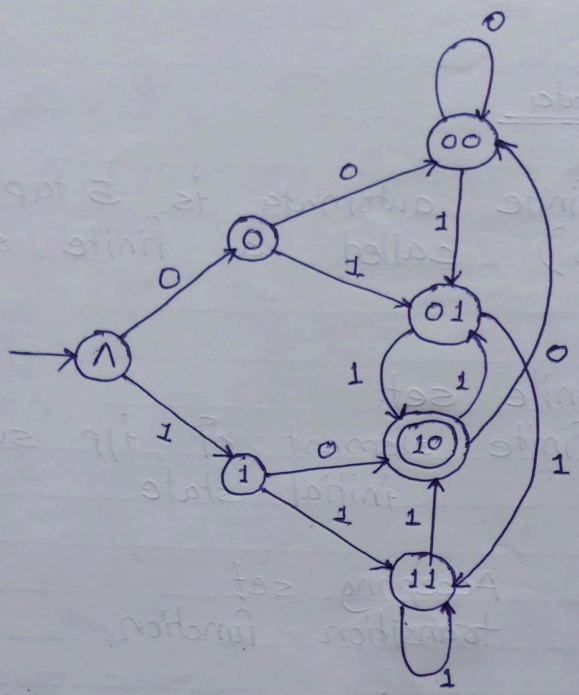
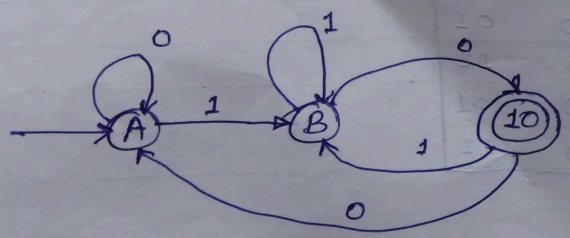


fig:- Transition Table

	Input	
	0	1
Λ	A	B
A	A	B
B	10	B
10	A	B

fig:- Transition table.



from the above

state	input 0	input 1
A	A	B
B	10	B
10	A	B

* Union, Intersections & Complements of Regular Language

Suppose L_1 & L_2 are both regular language over an alphebet Σ. There are FAs M_1 & M_2 accepting L_1 & L_2 respectively

- Then

Union $L_1 \cup L_2$

Concanation $L_1 L_2$

Kleene (closure) L_1^* are also regular language.

- If a string x belong to $L_1 \cup L_2$ then

$x \in L_1$ or $x \in L_2$ & gives two languages

$L_1 \cap L_2$, $L_1 - L_2$

- If

$M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$

$M_2 = (Q_2, \Sigma, q_2, A_2, \delta_2)$

Are the two automata of L_1 & L_2 then

$M = M_1 \cup M_2$

The ordered pair will be $Q_1 \times Q_2$

$\forall$ property of ordered pair.

The pair of initial state will be (q_1, q_2)

If M is in the state (p,q) ie p & q are current states of M_1 & M_2 & receives i/p symbol a

∴ two states will be

$$\boxed{(\delta_1(p,a), \delta_2(q,a))}$$

Suppose

$$M_1 = (Q_1, \Sigma_1, q_1, A_1, \delta_1) \text{ &}$$

$$M_2 = (Q_2, \Sigma_2, q_2, A_2, \delta_2)$$

accepts the language L_1 & L_2 respectively & M be the automata accepted by

$$M = (Q, \Sigma, q_0, A, \delta)$$

$$Q = Q_1 \times Q_2$$

$$q_0 = (q_1, q_2)$$

The transition func can be defined by

$$\delta((p,q),a) = (\delta_1(p,a), \delta_2(q,a))$$

For any
$P \in Q_1$
$q \in Q_2$
$a \in \Sigma$ then

1) If $A = \{(P,q) \mid P \in A_1 \text{ or } q \in A_2\}$,
M accepts language $L_1 \cup L_2$

2) If $A = \{(P,q) \mid P \in A_1 \text{ and } q \in A_2\}$
M accepts the language $L_1 \cap L_2$

3) If $A = \{(P,q) \mid P \in A_1 \ \& \ q \notin A_2\}$
M accepts the language $L_1 - L_2$

Proof -

Now let M be the finite automata with M_1 & M_2 with extended Transition funn δ^*, δ_1^* & δ_2^*

$$\delta^*((P,q),x) = (\delta_1^*(P,x), \delta_2^*(q,x))$$

which hold for any $x \in \Sigma^*$ & $(P,q) \in Q$.

By Mathemaheal induction

$(\delta_1^*(q_1,q_2),x) \in A$
iff x is accepted by M.

$(\delta_1^*(q_1, x), \delta_2^*(q_2, x)) \in A$

If we consider case 1 then we can get

$\delta_1^*(q_1, x) \in A_1$ or $\delta_2^*(q_2, x) \in A_2$

in other words

$$x \in L_1 \cup L_2$$

* Non Deterministic finite Automaton

A Non deterministic finite automata abbrivated NFA is 5 tuple $M = (Q, \Sigma, q_0, A, \delta)$ when Q & Σ are nonempty finite sets $q_0 \in Q$, A & $\delta : Q \times \Sigma \to 2^Q$.

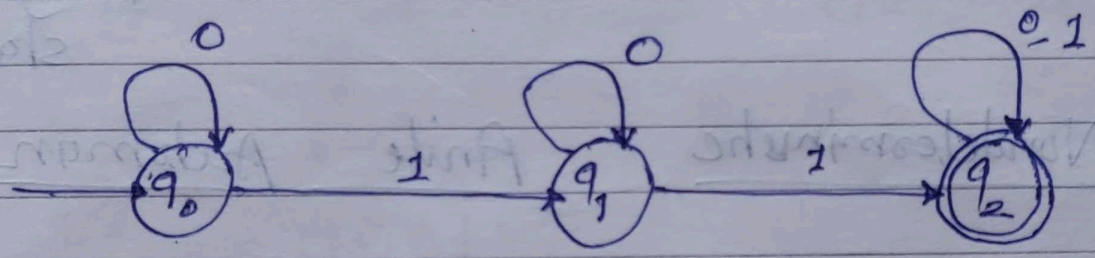
$\forall$ Q = set of states
Σ = Alphebet
q_0 = initial state
A = Accepting states

The power to be several states at once in deterministic automata only single power or single transaction

String Ending with 01

In Deterministic Finite Automata (DFA)

i) Only single transition fun will be there.



Transition diagram

ii). Instead of Accepting state only single transition

	0	1
q_0	q_0	q_1
q_1	q_1	q_2
q_2	q_2	q_2

Non Deterministic Automan

It will Contain two transitions or two arcs at a single state & Accepting state must of the time should be empty.

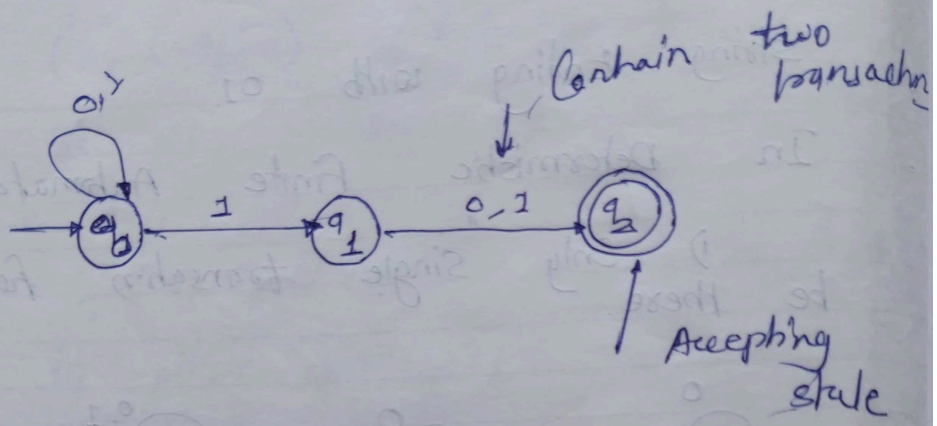
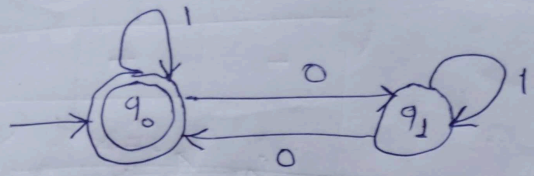


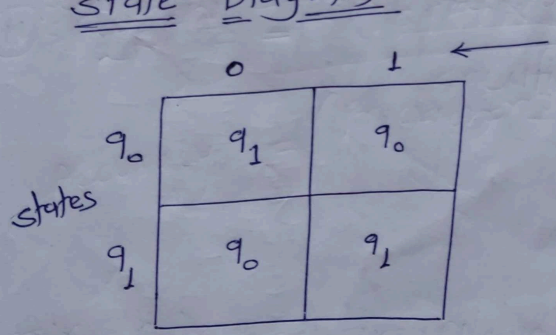
Fig:- Nondeterministic finite Automata

* Check Weather following is NFA / DFA



String	Acceptance / Reject
(a) 1	✓
(b) 101	X
(c) 1*	
(d) 11001	✓
(e) 10	
(f) 101011	
(g) 0*1*	
(h) 0	X
(i)	

State Diagram



i) $\delta(q_0, 0) = q_1$
ii) $\delta(q_0, 1) = q_0$
iii) $\delta(q_1, 0) = q_0$
iv) $\delta(q_1, 1) = q_1$

} Transactions

$1^* = \{\wedge, 1, 11, 111, 1111, \ldots\}$

(a) $1 \Rightarrow \delta(q_0, 1) = q_0$ ✓

(b) $11 \Rightarrow \delta(q_0, 1) = q_0$

$\delta(q_0, 1) = q_0$ ✓

$1^* = \{\wedge, 1, 11, 111, 1111, \ldots\}$

$= \{1, 11, 111, 1111, \ldots\}$

$0^* = \{\wedge, 0, 00, 000, 0000, \ldots\}$

$= \{0, 00, 000, 0000, \ldots\}$

(a) $0 \Rightarrow \delta(q_0, 0) = q_1$ ✗

(b) $00 \Rightarrow \delta(q_0, 0) = q_1$

$\delta(q_1, 0) = q_0$ ✓

(c) $000 \Rightarrow \delta(q_0, 0) = q_1$

$\delta(q_1, 0) = q_0$

$\delta(q_0, 0) = q_1$ ✗

(d) $000 \Rightarrow \delta(q_0, 0) = q_1$

$\delta(q_1, 0) = q_0$

$\delta(q_0, 0) = q_1$; $\delta(q_1, 0) = q_0$ ✓

① 101 ✗

$\delta(q_0, 1) = q_0$

$\delta(q_0, 0) = q_1$

$\delta(q_1, 1) = q_1$

② 11001 ✓

$-\ \delta(q_0, 1) = q_0$

$-\ \delta(q_0, 1) = q_0$

$-\ \delta(q_0, 0) = q_1$

$-\ \delta(q_1, 0) = q_0$

$\delta(q_0, 1) = q_0$

③ 0100111

$\delta(q_0, 0) = q_1$

$\delta(q_1, 1) = q_1$

$\delta(q_1, 0) = q_0$

$\delta(q_0, 0) = q_1$

$\delta(q_1, 1) = q_1$

$\delta(q_1, 1) = q_1$ ✓

$0^* 1^*$

$0^* = \{\Lambda, 0, 00, 000, \ldots\}$

$1^* = \{\Lambda, 1, 11, 111, \ldots\}$

$0^* 1^* = \{\Lambda, \Lambda 1, \Lambda 11, \Lambda 111, 0\Lambda, 01, 011, 0111, 00\Lambda, 001, 0011, 00111, 000\Lambda, 0001, 00011, 000111$

$= \{1, 11, 111, 0, 01, 011, 0111, 00, 001, 0011, 00111, 000, 0001, 00011, 000111, \ldots\}$

$0^* 1^* = \{0, 1, 00, 01, 11, \ldots\}$

(a) $0 \Rightarrow \delta(q_0, 0) = q_1$ ✗

(b) $1 \Rightarrow \delta(q_0, 1) = q_0$ ✓

(c) $00 \Rightarrow \delta(q_0, 0) = q_1$

$\delta(q_1, 0) = q_0$ ✓

(d) $0001 \Rightarrow \delta(q_0, 0) = q_1$

$\delta(q_1, 0) = q_0$

$\delta(q_0, 0) = q_1$

$\delta(q_1, 1) = q_1$ ✗

Hence it is not Accepted by FA.

Moore Machine :- o/p depends only on s

Mealy M/C :- o/p depends on i/p & s

* Draw the state diagram using following table & string accepted by FA 110001

state \ IP	0	1
→ q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

$\delta(q_0, 11000) = \delta(q_1, 1000)$

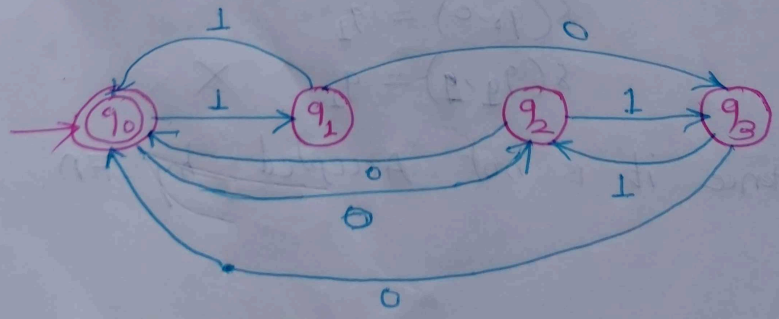
$= \delta(q_0, 000)$

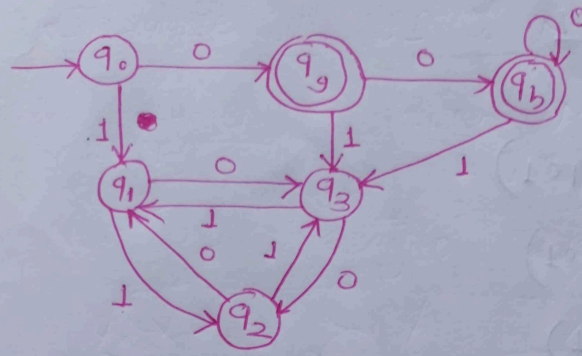
$= \delta(q_2, 00)$

$= \delta(q_3, 0)$

$= \delta(q_1, 1)$

$= q_0$ Accepting state





$q_0 \Rightarrow$ Initial state

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

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

$F = \{ q_g, q_h \}$

Transaction table

	0	1
→ q_0	q_g	q_1
q_1	q_3	q_2
q_2	q_1	q_3
q_3	q_2	q_1
(*) q_g	q_h	q_3
(*) q_h	q_h	q_3

(a) $S_1 = 10101$

$|S_1| = 5$

$\delta(q_0, 10101)$

$\delta(q_1, 0101)$

$\delta(q_3, 101)$

$\delta(q_2, 01)$

$\delta(q_1, 1) = q_2$ Non Final State

(b) $S_2 = 0^*11$

$= \{11, 011, 0011, 00011, \ldots\}$

$S_{12} = 11$

$\delta(q_0, 11)$

$\delta(q_1, 1) = q_2$ N Accepting

(c) $S_3 = 0^*$

$= \{0, 00, 000\}$

$\delta(q_0, 0) = q_0$ Accepted

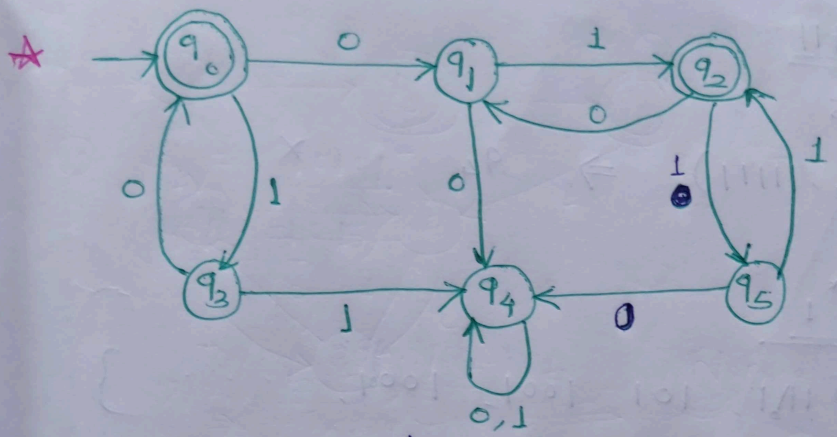


Fig:- State Diagram

	0	1
→ q_0	q_1	q_3
q_1	q_4	q_2
q_2	q_1	q_5
q_3	q_0	q_4
q_4	q_4	q_4
q_5	q_4	q_2

Fig:- Transaction Table

① 11111

$|S_1| = 5$

$\delta(q_0, 11111) \Rightarrow q_4$ ✗

② $1 0^* 1$

$S_2 = \{11, 101, 1001, 10001, \ldots\}$

$= \{11, 101, 1001, 1001, \ldots\}$ ✗

③ 0101

$|S_3| = 4$

$\delta(q_0, 0101) = q_2$ ✓

$\delta(q_0, 0101)$

$\delta(q_1, 101)$

$\delta(q_2, 01)$

$\delta(q_1, 1) = q_2$ Accepting state

Applications of Finite Automata

Applications of finite automata include string matching algorithms, network protocols and lexical analyzers

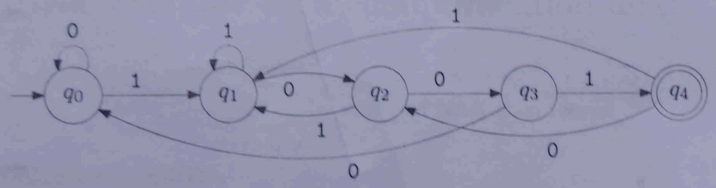
String Processing

Consider finding all occurrences of a short string (***pattern string***) within a long string (***text string***).

This can be done by processing the text through a DFA: the DFA for all strings that ***end*** with the pattern string. Each time the accept state is reached, the current position in the text is output.

Example: Finding 1001

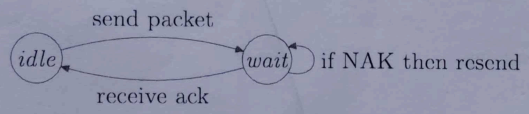
To find all occurrences of pattern 1001, construct the DFA for all strings ending in 1001.



Finite-State Machines

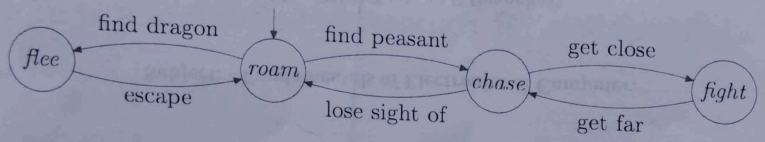
A ***finite-state machine*** is an FA together with actions on the arcs.

A trivial example for a communication link:



Example FSM: Bot Behavior

A ***bot*** is a computer-generated character in a video game.

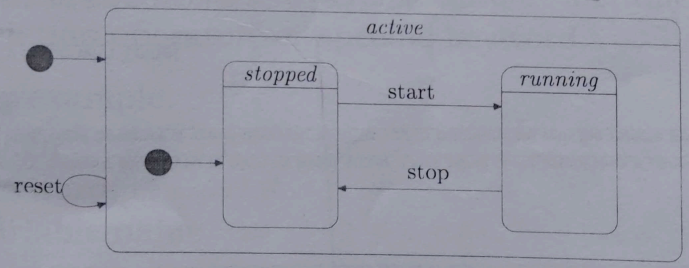


Note that using finite-state machine allows automation.

Statecharts

Statecharts model tasks as a set of states and actions. They extend FA diagrams.

Here is a simplified statechart for a stopwatch.



Lexical Analysis

In compiling a program, the first step is ***lexical analysis***. This isolates keywords, identifiers etc., while eliminating irrelevant symbols.

A ***token*** is a category, for example "identifier", "relation operator" or specific keyword.

For example,

token	*RE*
keyword then	`then`
variable name	`[a-zA-Z][a-zA-Z0-9]*`

where latter RE says it is any string of alphanumeric characters starting with a letter.

Lexical Analyzer

A lexical analyzer takes source code as a string, and outputs sequence of ***tokens***.

For example,

```
for i = 1 to max do
    x[i] = 0;
```

might have token sequence

for | id | = | num | to | id | do | id | [| id |] | = | num | sep

As a token is identified, there may be an action. For example, when a number is identified, its value is calculated,

Applications of Deterministic Finite Automata

Eric Gribkoff
ECS 120
UC Davis

Spring 2013

1 Deterministic Finite Automata

Deterministic Finite Automata, or DFAs, have a rich background in terms of the mathematical theory underlying their development and use. This theoretical foundation is the main emphasis of ECS 120's coverage of DFAs. However, this handout will focus on examining real-world applications of DFAs to gain an appreciation of the usefulness of this theoretical concept. DFA uses include protocol analysis, text parsing, video game character behavior, security analysis, CPU control units, natural language processing, and speech recognition. Additionally, many simple (and not so simple) mechanical devices are frequently designed and implemented using DFAs, such as elevators, vending machines, and traffic-sensitive traffic lights.

As the examples below will demonstrate, DFAs naturally lend themselves to concisely representing any system which must maintain an internal definition of state. Our examples begin with vending machines, which need to remember how much money the user has input, and continue to more complicated examples of video game agent AI and communication protocols. As our final example, we will consider the incorporation of finite state machines into the Apache Lucene open-source search engine, where they are used to implement search term auto-completion.

Formally, a deterministic finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ such that:

1. Q is a finite set called the **states**
2. Σ is a finite set called the **alphabet**
3. $\delta : Q \times \Sigma \rightarrow Q$ is the **transition function**

4. $q_0 \in Q$ is the **start state**

5. $F \subseteq Q$ is the **set of accept states**

We also discuss Mealy machines, which add to the expressiveness of DFAs by producing output values at each transition, where the output depends on the current state and input. Rather than accepting or rejecting strings, Mealy machines map an input string to an output string. They can be thought of as functions that receive a string and produce a string in response.

Formally, a Mealey machine is a 6-tuple $(Q, \Sigma, \Gamma, \delta, \omega, q_0)$ such that:

1. Q, Σ, δ, and q_0 are defined as in a DFA.

2. Γ is a finite set called the **output alphabet**

3. $\omega : Q \times \Sigma \to \Gamma$ is the **output function**

2 A Non-Exhaustive List of DFA Applications

- Vending Machines
- Traffic Lights
- Video Games
- Text Parsing
- Regular Expression Matching
- CPU Controllers
- Protocol Analysis
- Natural Language Processing
- Speech Recognition
- Lexical Analysis
- Pattern Recognizing
- Design Sequential Circuit
- Implement Spell checkers

3 Vending Machines

Figure 1 presents a DFA that describes the behavior of a vending machine which accepts dollars and quarters, and charges \$1.25 per soda. Once the machine receives at least \$1.25, corresponding to the blue-colored states in the diagram, it will allow the user to select a soda. Self-loops represent ignored input: the machine will not dispense a soda until at least \$1.25 has been deposited, and it will not accept more money once it has already received greater than or equal to \$1.25.

To express the DFA as a 5-tuple, the components are defined as follows:

1. $Q = \{\$0.00, \$0.25, \$0.50, \$0.75, \$1.00, \$1.25, \$1.50, \$1.75, \$2.00\}$ are the **states**
2. $\Sigma = \{\$0.25, \$1.00, \textit{select}\}$ is the **alphabet**
3. δ, the **transition function**, is described by the state diagram.
4. $q_0 = \$0.00$ is the **start state**
5. $F = \emptyset$ is the **set of accept states**

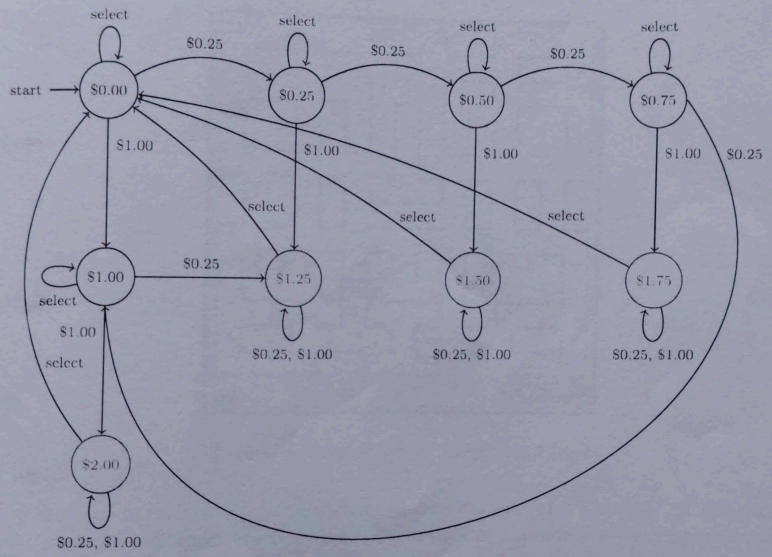


Figure 1: Vending Machine State Diagram

4 AI in Video Games: Pac-Man's Ghosts

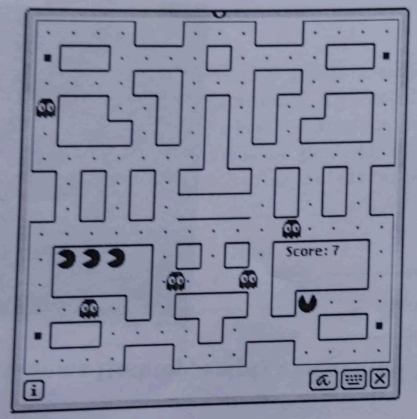


Figure 2: Screenshot of a Pacman Clone

Finite state machines lend themselves to representing the behavior of computer-controller characters in video games. The states of the machine correspond to the character's behaviors, which change according to various events. These changes are modeled by transitions in the state diagram. State machines are certainly not the most sophisticated means of implementing artificially intelligent agents in games, but many games include characters with simple, state-based behaviors that are easily and effectively modeled using state machines.

Here we consider the classic game, Pac-Man. For those unfamiliar with the game-play, Pac-Man requires the player to navigate through a maze, eating pellets and avoiding the ghosts who chase him through the maze. Occasionally, Pac-Man can turn the tables on his pursuers by eating a power pellet, which temporarily grants him the power to eat the ghosts. When this occurs, the ghosts' behavior changes, and instead of chasing Pac-Man they try to avoid him.

The ghosts in Pac-Man have four behaviors:

1. Randomly wander the maze

2. Chase Pac-Man, when he is within line of sight

3. Flee Pac-Man, after Pac-Man has consumed a power pellet

4. Return to the central base to regenerate

These four behaviors correspond directly to a four-state DFA. Transitions are dictated by the situation in the game. For instance, a ghost DFA in state 2 (Chase Pac-Man) will transition to state 3 (Flee) when Pac-Man consumes a power pellet.

For a further discussion of state machines for game AI, see `http://research.ncl.ac.uk/game/mastersdegree/gametechnologies/aifinitestatemachines/`.

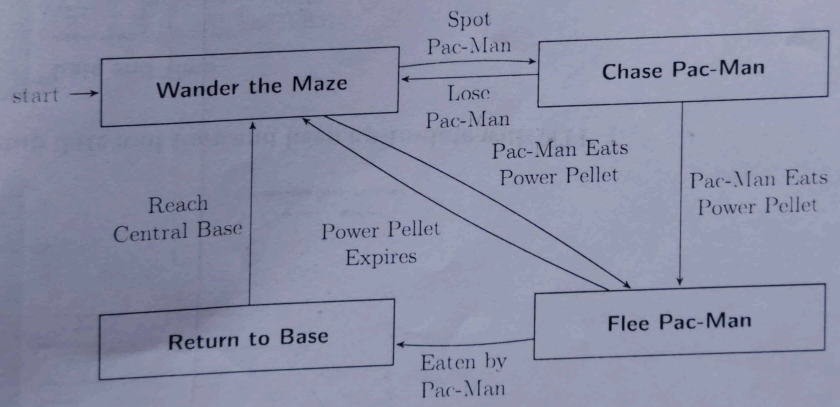


Figure 3: Behavior of a Pac-Man Ghost

5 Internet Protocols: TCP as a DFA

Internet protocols also lend themselves to descriptions as DFAs. The state diagram below represents a simplified version of the Transmission Control Protocol (TCP).

The state machine in Figure 4 describes the "life stages" of a TCP connection. A connection between two computers begins in the closed state. Following a series of signals, described by the transition arcs of the diagram, the two machines reach the established state where communication can proceed. Once completed, the transition arcs describe the process whereby the connection returns to the closed state.

See `http://www.tcpipguide.com/free/` for further discussion of the TCP protocol, and `http://www.texample.net/tikz/examples/tcp-state-machine/` for the source used to generate the state machine diagram.

Representation of complex protocols, such as TCP or Transport Layer Security (TLS), as DFAs helps in understanding the protocols and aids implementations, as the state diagrams clearly illustrate allowed and disallowed transitions between states. In addition, finite state security analysis has been used to examine the security of protocols such as SSL and TLS.

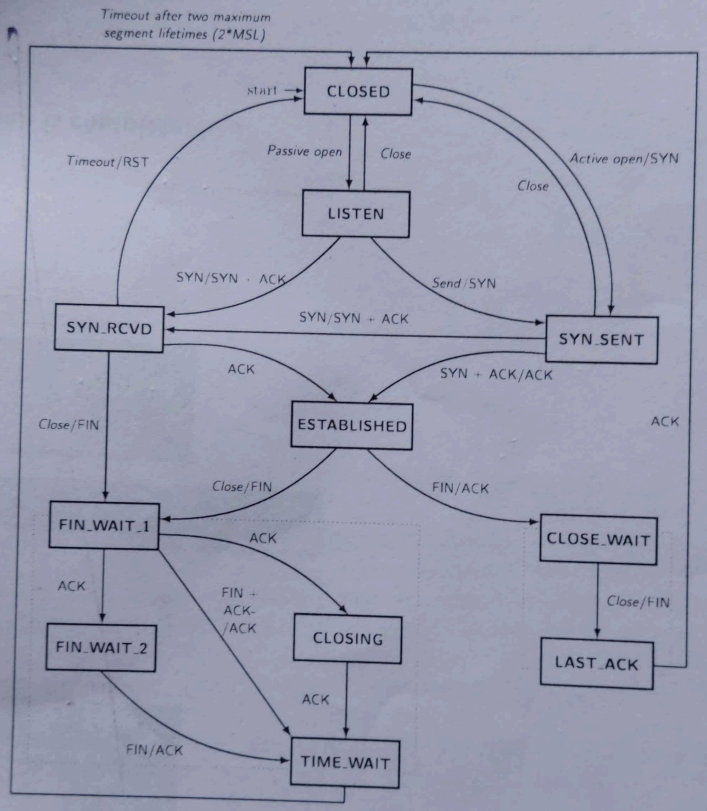


Figure 4: TCP State Diagram

6 Auto-Complete in Apache Lucene

Simple extensions to the DFA definition can greatly expand their power while preserving their ability to concisely encapsulate ideas and processes. One such extension is the finite state transducer (FST), which enhances the DFA definition by adding an output capability. Mealy machines, defined above, are a type of FST.

The figure below illustrates the use of a Mealy machine to index strings in a sorted array. Words are matched to their index in the search term data structure by processing the words through the Mealy machine and summing the output values encountered. (Empty outputs are omitted from the diagram.)

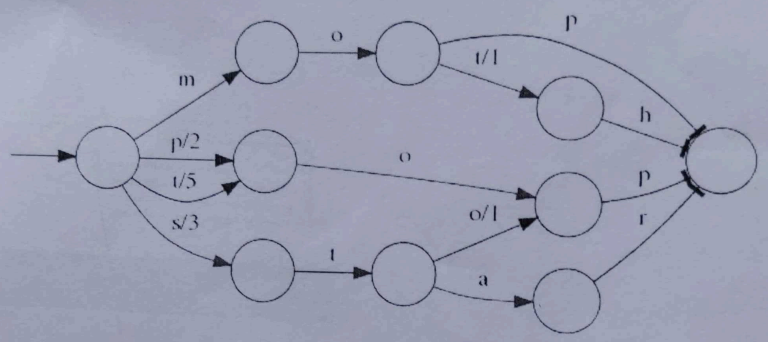


Figure 5: FST for Text Lookups

This Mealy machine proceses the words in the ordered set $\{mop, moth, pop, star, stop, top\}$ to produce the numbers corresponding to their alphabetical order within the set. Consider *stop*, the fifth (index = 4) element of the set. Beginning with the start state, we follow the *s* arc. This arc has associated output 3. Then we follow the *t* arc, with no output. Then we follow the *o* arc, with output 1. Our sum currently stands at four. Then we follow the *p* arc, with no output, and reach the end of the word. Our sum gives us the index of *stop*: 4.

The advantage of the FST approach is that the common prefixes and suffixes of stored terms are shared, greatly reducing the memory footprint required for text lookups.

The Apache Lucene project used a Mealy machine to dramatically decrease the memory footprint of their auto-complete search term feature. By using an FST to conduct lookups in its search term index, the Apache Lucene project is able to store the entire index in memory, resulting in much faster lookups. More information on the approach adopted by the Lucene project can be found at `http://blog.mikemccandless.com/2010/12/using-finite-state-transducers-in.html`.