Unit V

Turing Machine

Ex. $I = \{0, 1, b\}$

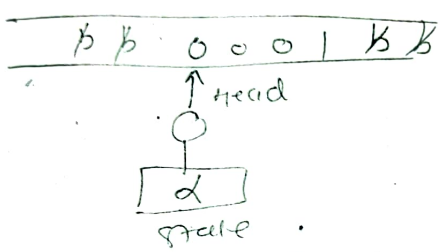
$S = \{\alpha, \beta, \gamma = \text{halt}\}$

$D = \{L, R, N\}$

SFM:

S \ I	0	1	b
α	R	0βR	R
β	-	-	γN
γ	-	-	-

Whenever sequence of 0's is followed by a '1' & blank is encountered, m/c will replace last or ending '1' by '0' & move to next avail b & halt.



Initial configuration of TM.

Simulation : i/p "0001"

b 0 0 0 1 b , b 0 0 0 1 b $\quad \delta(\alpha, 0) \rightarrow R.$

α ① , α α α ② ③ ④

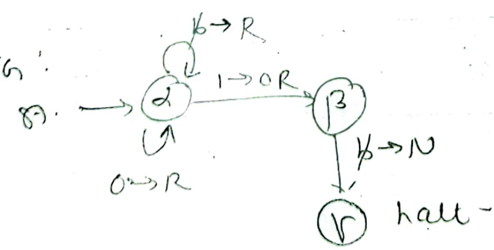
b 0 0 0 0 b $\quad \delta(\alpha, 1) \rightarrow 0\beta R.$

β

b 0 0 0 0 b $\quad \delta(\beta, b) \rightarrow \gamma N.$

γ(halt)

TG:



Ex. Well formedness of parenthesis.

	(	)	*	B
q_0 look for) bracket	q_0 R (skip all (, move right for)	q_1 L * change) to *, move left for (	q_0 R * skip all *, move right for)	q_2 L B. move left to validate
q_1 look for (	q_0 R * change (to *, right for)	not possible	q_1 L * skip all *, move left	Error extra).
q_2 validate	Error extra (	not possible	q_2 L * skip *, move left till B	Accept all are *.

① (()(())

(**())

(****)

② B)) ((error in q_1

q_1 q_0

③ B(()B error in q_2

q_0

(**

q_2

④ ()()) error in q_1

⑤ ((((error in q_2

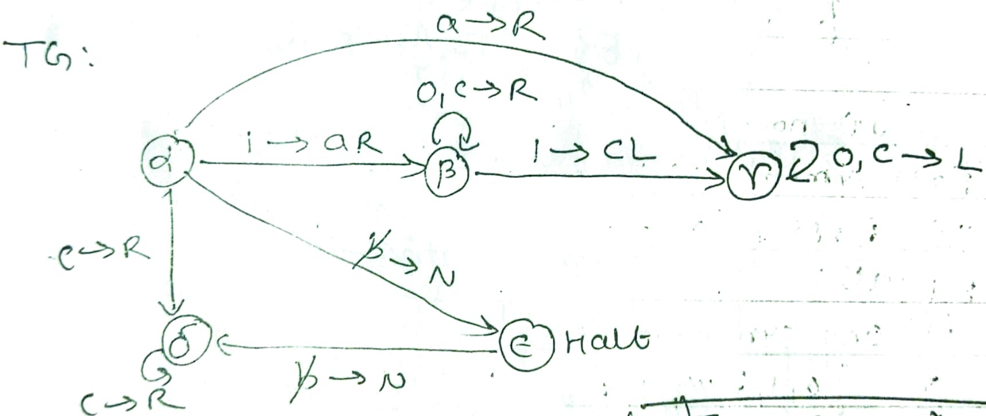
Ex. $0^n 1^n$.

Replace 0 by a, move right, replace 1 by c, move left till a. move right one pos^n.
Repeat.

$I = \{0, 1, a, c, b\!\!\!/\}$
$S = \{\alpha, \beta, \gamma, \delta, \epsilon = \text{halt}\}$.
$D = \{L, R, N\}$.
α - initial state, ϵ - halt state.

S \ I	0	1	a	c	$b\!\!\!/$
α	$a\beta R$	err	–	δR	ϵN accept
β	R	$c\gamma L$	–	R	err
γ	L	–	αR	L	–
δ	–	–	–	R	ϵN (accept)
ϵ	–	–	–	–	–

TG:



Simulation:

$b\!\!\!/ \, 0 \, 0 \, 1 \, 1 \, b\!\!\!/$ - initial conf.
↑ α

a 0 1 1 $\quad \delta(\alpha, 0) = (a, \beta, R)$
↑ β

a 0 1 1 $\quad \delta(\beta, 0) = R$
↑ β

a 0 c 1 $\quad \delta(\beta, 1) = (c, \gamma, L)$
↑ γ

a 0 c 1 $\quad \delta(\gamma, 0) = L$
↑ γ

S \ I	0	1	*	B
q_0	$q_1 * R$	err	$q_0 * L$	$q_2 B R$
q_1	$q_1 0 R$	$q_0 * L$	$q_1 * R$	err
q_2	err	err	$q_2 * R$	$q_3 N$ accept
q_3	–	–	–	–

a 0 c 1
↑
α
$\delta(\alpha, 0) = (a, \beta, R)$

a a c 1
↑
β
$\delta(\beta, c) = R$

a a c 1
↑
β
$\delta(\beta, 1) = (c, \gamma, L)$

a a c c
↑
γ
$\delta(\gamma, c) = L$

a a c c
↑
γ
$\delta(\gamma, a) = (\alpha, R)$

a a c c
↑
α
$\delta(\alpha, c) = (\delta, R)$

a a c c
↑
δ
$\delta(\delta, c) = R$

a a c c b
↑
δ
$\delta(\delta, b) = \epsilon N.$

a a c c b
↑
ε

Ex. Equal no. of 0's & 1's:
first symbol (0|1) replace it by *, move right
till first (1|0), " " *
Repeat.
if any symbol remains, error.
$I = \{0, 1, ;, *, T, F\}$ (T = Accepted, F = Rejected)
$S = \{q_0, q_1, q_2, q_3, q_4 = \text{halt}\}$
$D = \{L, R, N\}$.

SFM: Simplified functional matrix –

S \ I	0	1	;	*	T	F
q_0	$*q_1R$	$*q_3R$	Tq_4N	R	–	–
q_1	R	$*q_2L$	Fq_4N	R	–	–
q_2	L	L	q_0R	L	–	–
q_3	$*q_2L$	R	Fq_4N	R	–	–
q_4	–	–	–	–	–	–

Simulation:

```
1 1 0 1  0 0
↑

* 1 0 1 0 0
  ↑ ↑

* 1 * 1 0 0
  ↑

* 1 * 1 0 0
↑

  1 * 1 0 0
```

```
; * 1 * 1 0 0
  ↑ ↑

* * * 1 0 0
    ↑ ↑ ↑

; * * * 1 * 0
↑ ↑ ↑ ↑ ↑ ↑

——————→ * * * *
        ↑     T
```

TG:

Ex. Binary Palindrome

Head pointing to ; before the actual symbol.
Read first symbol (0/1) replace with ;
move to right end.
Read last symbol, replace with ; if it is equal to symbol which we replaced at other end.
If not equal, then not palindrome.

$I = \{0, 1, ;, F, T\}$.
$S = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6 = \text{halt}\}$.

SFM:

	S \ I	0	1	;	F	T
left end	q_0	L	L	q_1R	—	—
1st symbol	q_1	; q_2R	; q_4R	T q_6N		
0-end	q_2	R	R	q_3L	—	—
matching symbol	q_3	; q_0L	F q_6N	T q_6N	—	—
1-end	q_4	R	R	q_5L		
match	q_5	F q_6N	; q_0L	T q_6N	—	—
	q_6	—	—	—	—	—

Ex. Add 2 unary Nos.

b̸ a a a a c a a b̸

delimiter.

Replace a in 1st No. by b̸ & append a after right hand No.

$I = \{a, b̸, c\}$

$S = \{\alpha, \beta, \gamma, \delta = \text{halt}\}$

s \ I		a	b̸	c
check a, replace b̸.	α	b̸ β R	R	b̸ δ N
append a	β	R	a γ N	R
Reset Head.	γ	L	α R	L.

Ex- Multiplication of 2 unary Nos.

both Nos. are represented using letter 1, separated by ,.

Remaining tape is filled with 0's.

Answer is written after the ending ;, using letter b.

Initial config:

multiplier multiplicand

0	;	1	1	,	1	1	;	0	0	0	0

R/W head.

state α

Final config:

Result

0	;	0	1	,	1	1	;	b	b	b	b

Logic - repetitive addition of multiplicand to itself.

If m is multiplier & n is multiplicand then

$n \times m = n + n + \ldots m$ No. of times.

Replace one '1' of multiplier by 0, (reduce by 1), add multiplicand to right end by appending n No. of b's to right end.

Repeat till all 1's in m get replaced by all 0's.

$I = \{0, 1, a, b, ;, ,\}$
$S = \{\alpha, \beta, \gamma, \delta, \epsilon, \phi = halt\}$

SFM:

	0	1	a	b	;	,
α	L	$0\beta R$	–	–	ϕN	L
β	R	aR	–	–	γL	R
γ	–	–	$1\delta R$	–	R	L
δ	$b\epsilon L$	R	–	R	R	–
ϵ	L	L	$1\delta R$	L	L	αN
ϕ	–	–	–	–	–	–

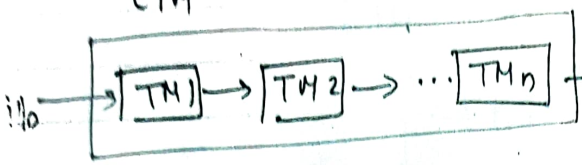
OR q_{-1} – replace b̸ at right end by *.
q_0 – replace 1 by b̸ & goto q_1
q_1 – goto right till u get c, move R, goto q_2
q_2 – if 1, replace by P; goto q_3
if *, goto q_5.
q_3 – keep going to R till b̸ & replace it by 1 & goto q_4.
q_4 – goto left till P, move to its R & goto q_2 (looping)
q_5 – keep going to L, & replace all P's by 1's till b̸, on b̸ move to right & goto q_0 (looping)

	1	c	b̸	P	*
q_{-1}	R	R	*, q_5	–	–
q_0	b̸R q_1	accept	–	–	–
q_1	R	R, q_2	–	–	–
q_2	P, q_3	–	–	–	q_5
q_3	R	R	1; q_4	R	–
q_4	L	–	–	R, q_2	–
q_5	L	L	R, q_0	1, L	L

Complexity of TM =
No. of symbols in I × No. of states
$|I| \times |S|$.

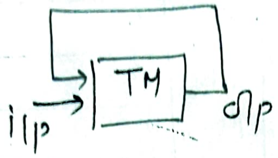
Composite TM & Iterated TM –

CTM



Solves collection of simpler problem.

ITM



Recursion

Used in modular programming –
divide ~~by~~ & conquer strategy

Universal TM (UTM) –
capable of doing anything that any other TM can do. It can imitate any TM T given foll. info. on its tape:
① Description of T – its FM (program area)
② Initial config. of T – (state area)
i.e. start state & symbol.
③ Processing data – i/p (Data area).

UTM should have imitation algo. to interpret correctly rules of operation given in FM of T.
It should have table look-up facility.
UTM performs foll:

Imitation Algo:
① Scan state area & read start state & symbol.
② Move tape to prog. area containing FM of T.

area, Replace symbol by new symbol, move head in reqd. dir^n, read next symbol & finally reach state area, replace state & scanned symbol.
goto step 1.

We need FM for UTM. 2D FM must be converted to 1D. Alphabet set I includes internal states of T as well.

ex. FM for TM 'T' is :

S\I	0	1	b̸
α	1βR	1αR	b̸αR
β	* * *	* * *	b̸γN

① 1D FM :

0α1βR0β* * * 1α1αR1β* * * b̸αb̸αRb̸βb̸γN.

I for UTM = 0, 1, b̸, α, β, γ, L, R, N, *

Gr. of 5 symbols — row, column, triplet.

② Encode these m symbols using binary code with n bits, $2^n \geq m$

Here $m = 10$, $n = 4$ $2^4 > 10 > 2^3$

0 = 0000, 1 = 0001, b = 0010,
α = 0011, β = 0100, γ = 0101,
L = 0110, R = 0111, N = 1000,
* = 1001.

③ After FM of T has been encoded, express the imitation algo. as FM of UTM. This UTM then solves same problem as T.

UTM is foundation for —
① Stored - program computers
② Interpretive implement^n of prog. lang.

Multistack TM

Symbols to left of head of TM can be stored on one 1 stack while sym. of rt. on other stack.

On each stack, symbols closer to the TM's head are placed closer to stack top.

Solvability, semi-solvability, Unsolvability

1. If there is a TM which when applied to any problem, always eventually terminates with correct yes or no answers, the problem is solvable.

2. repeat (·), correct answer when ans. is yes & may or may not terminate when correct answer is no, problem is semi- or partially solvable.

3. If there is no TM which when applied to a problem eventually terminates with correct answer "yes", problem is unsolvable.

Halting Problem -

For a given config. of TM, 2 cases arise:

1. m/c starting at this config. will halt after a finite No. of steps.
2. m/c starting at this config., never halts no matter how long it runs.

Given any TM, problem of determining whether it halts ever or not, is called halting problem.

To solve halting prob., given any TM, i/p data tape & init. config., we should have mechanism to determine whether process will ever halt or not.

In reality one can not solve halting prob.; it is unsolvable.
∴ No TM (Prog.) can detect whether a given TM (Prog.) will ever halt or not.

Recursively Enumerable & Recursive Sets.

Lang. accepted by a TM is recursively enumerable.

Recursively enumerable set – (r.e.)

A set 'S' of words over Σ is r.e. if there is a TM over Σ which accepts every word in S & either rejects or loops for every word in ($\Sigma^* - S$) i.e.

accept (TM) = S

reject (TM) U loop (TM) = $\Sigma^* - S$.

Recursive set – S

if TM accepts every word in S & rejects every word in (–S).

loop (TM) = ϕ

Functions – TM may be viewed as a comp^n of f^ns from int. to int., represented in unary.

Total recursive f^n –

if $f(i_1, i_2, \ldots i_k)$ is defined for all i_1 to i_k then f is tot. rec. f^n. They correspond to rec. lang., ∵ they are computed by TM that always halts.

ex. all common arithmatic f^ns on int.

multiplic^n, n!, $\log_2 n$, 2^{2^n}

partial rec. f^n – f^n may or may not halt on given i/p. correspond to recursively enumerable lang., ∵ they are computed by TM that halts on acceptance but for rejected i/p may or may not halt.