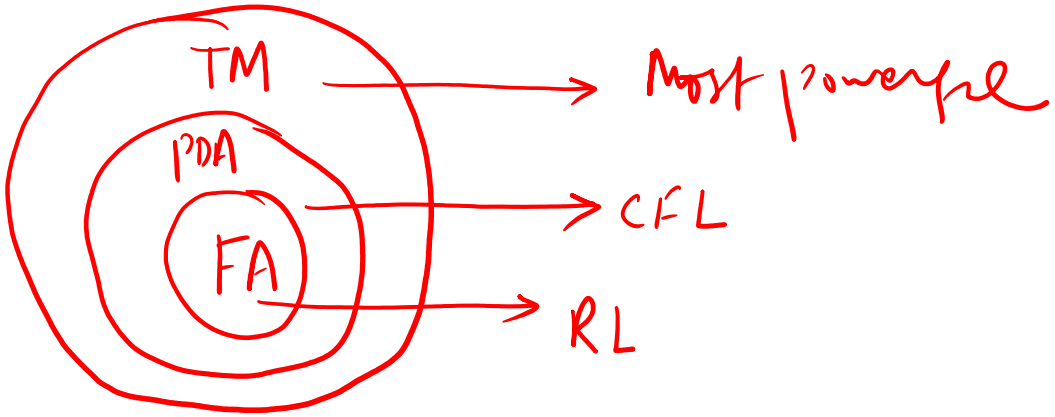


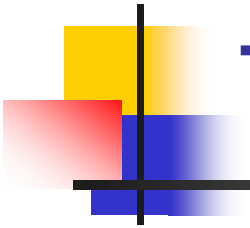
Turing Machines

Alan Turing 1936

By Prashant Gautam

e.g. $a^n b^n \mid n \geq 1$ — PDA ✓ FA ✗ TM ✓

$a^n b^n c^n \mid n \geq 1$ — PDA ✗ FA ✗ TM ✓ (powerful)



Turing Machines are…

- Very powerful (abstract) machines that could simulate any modern day computer (although very, very slowly!)

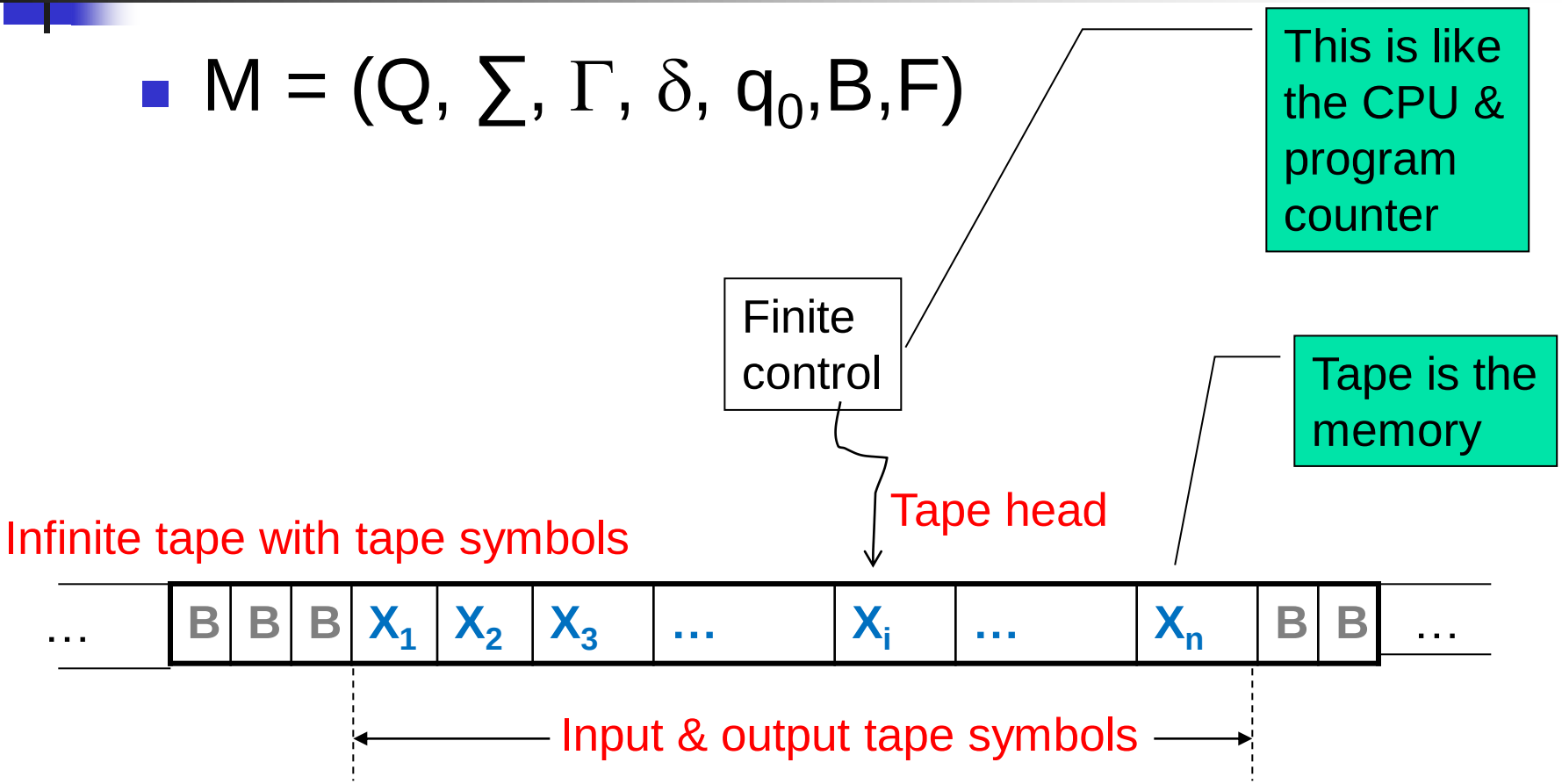
- Why design such a machine?
  - If a problem cannot be "solved" even using a TM, then it implies that the problem is ***undecidable***

For every input, answer YES or NO

- Computability vs. Decidability

A Turing Machine (TM)

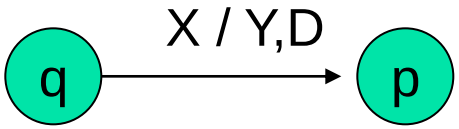
- $M = (Q, \sum, \Gamma, \delta, q_0, B, F)$



You can also use:
➔ for R
🡐 for L

Transition function

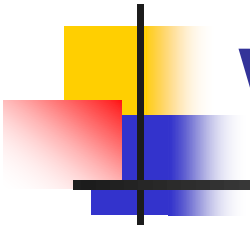
- One move (denoted by |---) in a TM does the following:
  - ***δ(q,X) = (p,Y,D)***
    - q is the current state
    - X is the current tape symbol pointed by tape head
    - State changes from q to p
    - <u>After the move:</u>
      - X is replaced with symbol Y
      - If D="L", the tape head moves "left" by one position.
        Alternatively, if D="R" the tape head moves "right" by one position.



ID of a TM

- Instantaneous Description or ID :
  - $X_1X_2\ldots X_{i-1}qX_iX_{i+1}\ldots X_n$
    means:
    - q is the current state
    - Tape head is pointing to X_i
    - $X_1X_2\ldots X_{i-1}X_iX_{i+1}\ldots X_n$ are the current tape symbols

- $\delta(q,X_i) = (p,Y,R)$ is same as:
  $X_1\ldots X_{i-1}qX_i\ldots X_n$ |---- $X_1\ldots X_{i-1}YpX_{i+1}\ldots X_n$
- $\delta(q,X_i) = (p,Y,L)$ is same as:
  $X_1\ldots X_{i-1}qX_i\ldots X_n$ |---- $X_1\ldots pX_{i-1}YX_{i+1}\ldots X_n$



Way to check for Membership

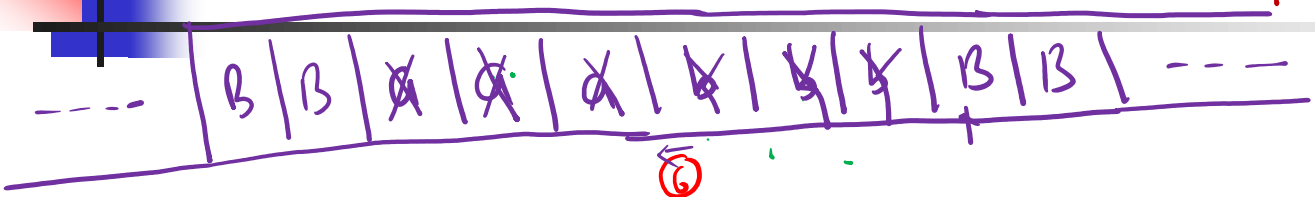
- Is a string *w* accepted by a TM?

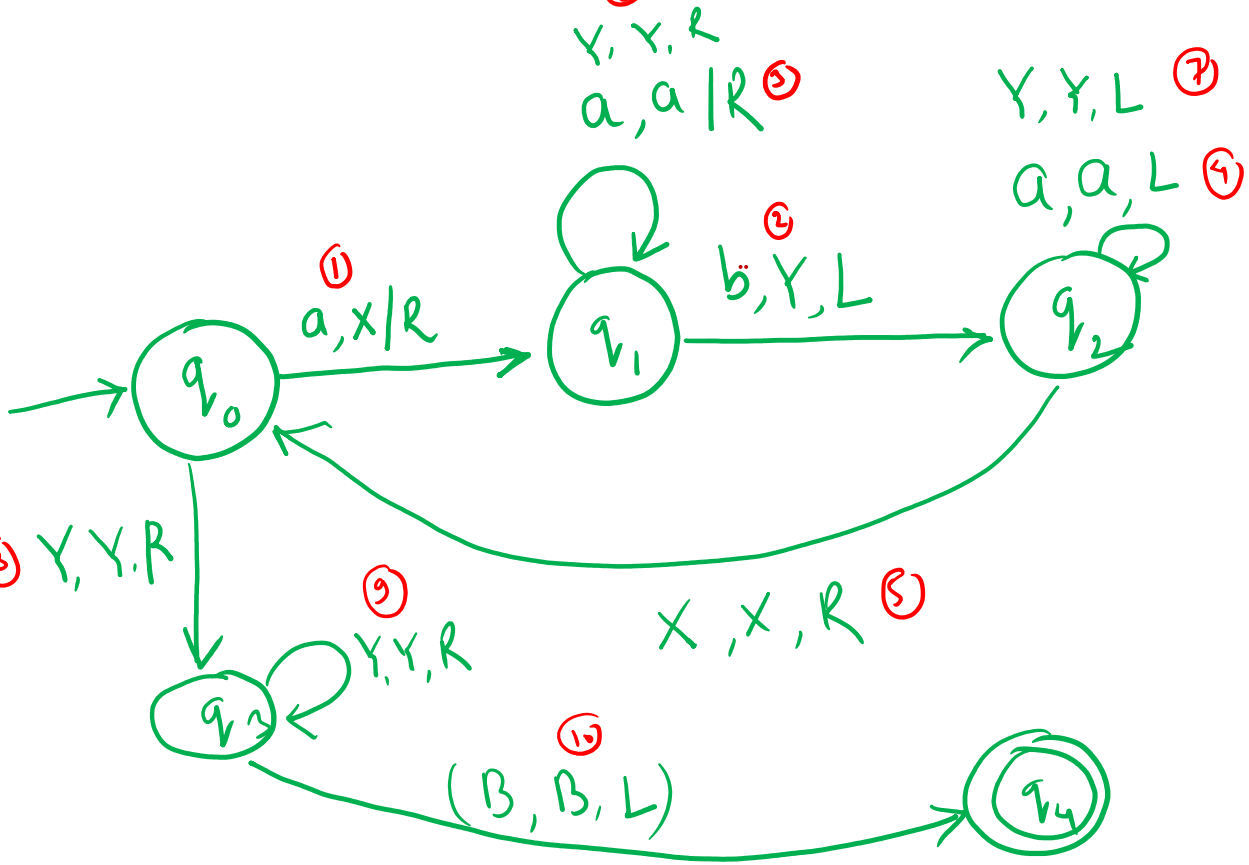
- Initial condition:
  - The (whole) input string *w* is present in TM, preceded and followed by infinite blank symbols
- Final acceptance:
  - Accept *w* if TM enters final state and halts
  - If TM halts and not final state, then reject

Example: Design a TM that accepts a Lang.

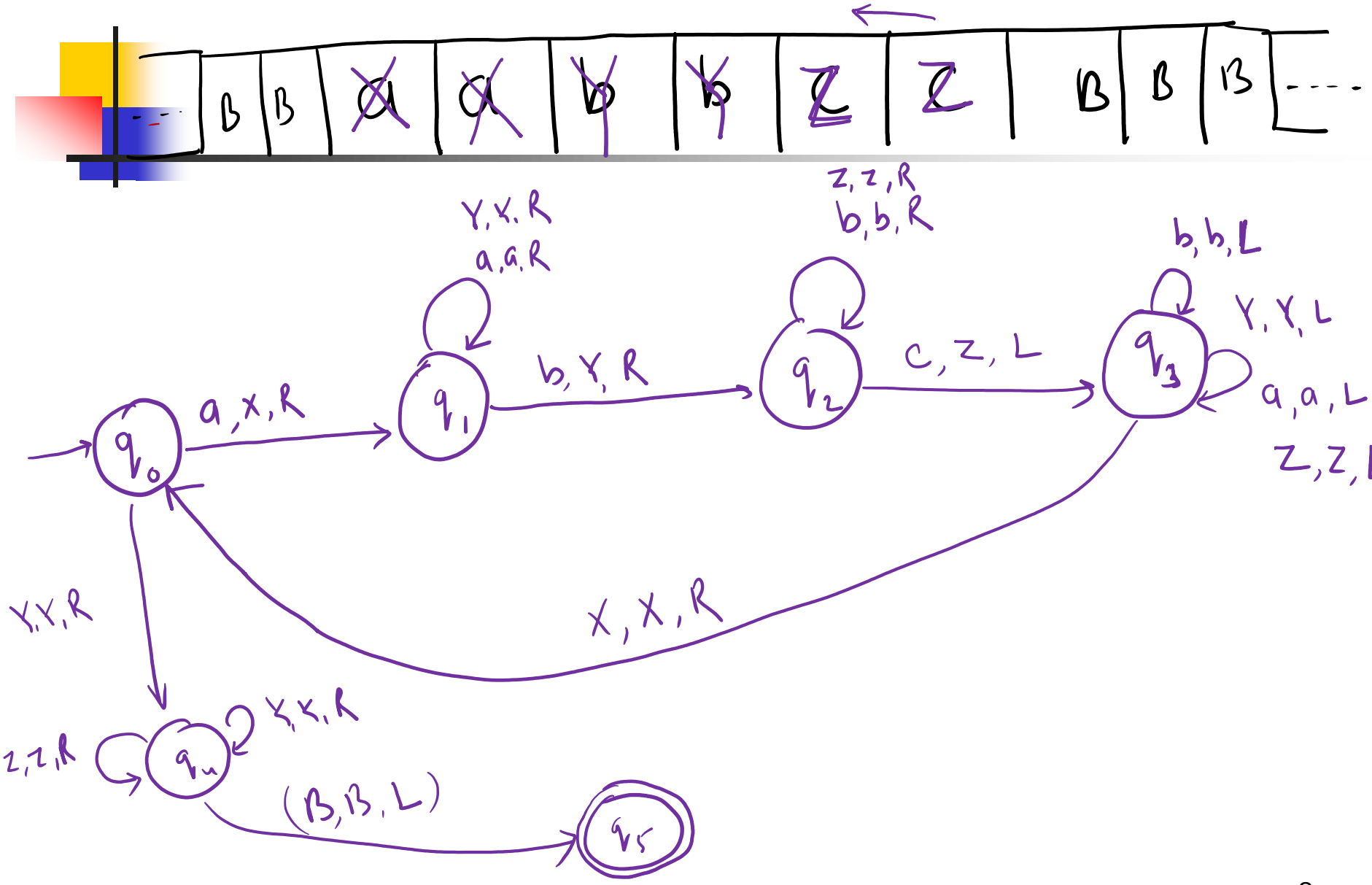
$a^n b^n \mid n >= 1.$

$\delta(q_0, X_i) = (P, Y, D)$



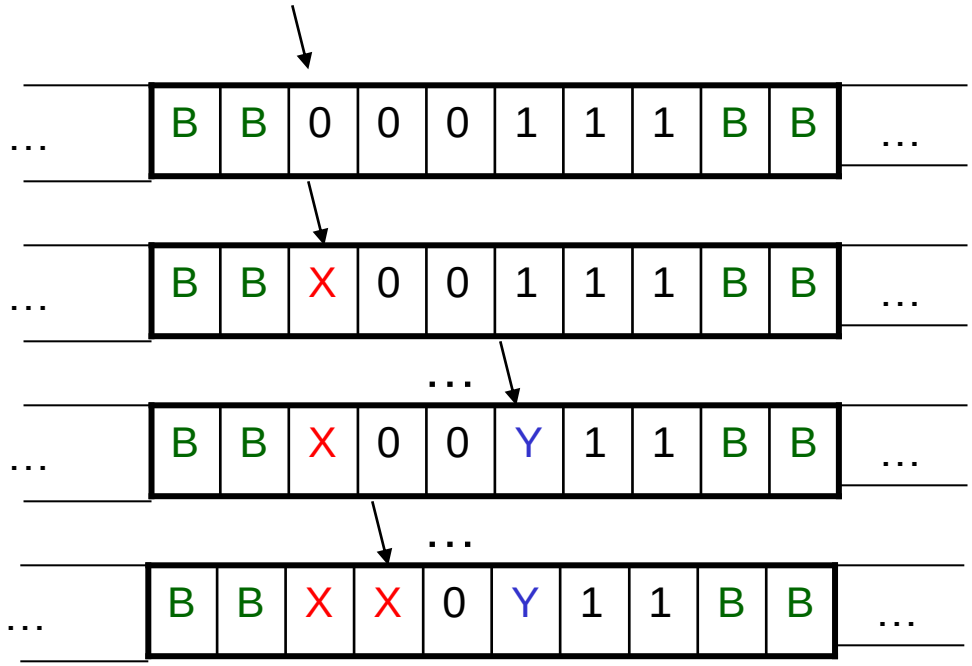


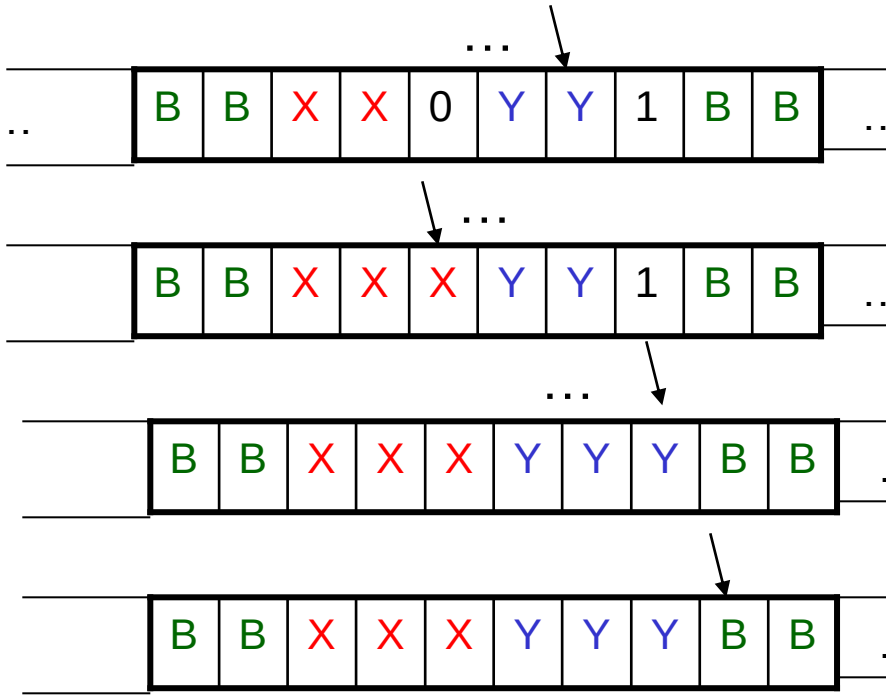
TM: $a^n b^n c^n \mid n >= 1$



Example: L = {0^n1^n | n≥1}

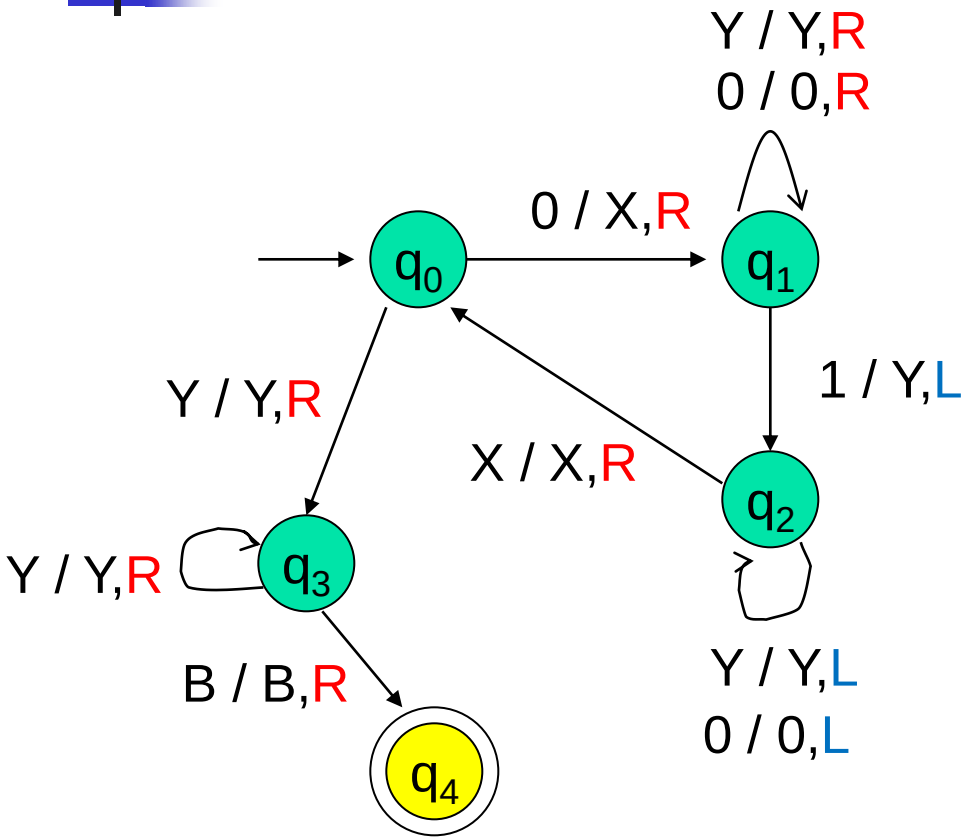
- Strategy: w = 000111





Accept

TM for $\{0^n1^n \mid n \geq 1\}$



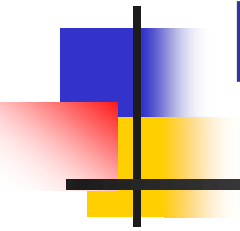
1. Mark next unread 0 with X and move right
2. Move to the right all the way to the first unread 1, and mark it with Y
3. Move back (to the left) all the way to the last marked X, and then move one position to the right
4. If the next position is 0, then goto step 1.
   Else move all the way to the right to ensure there are no excess 1s. If not move right to the next blank symbol and stop & accept.

*state diagram representation preferred

TM for $\{0^n1^n \mid n \geq 1\}$

	Next Tape Symbol				
Curr. State	0	1	X	Y	B
→ q_0	(q_1,X,R)	-	-	(q_3,Y,R)	-
q_1	$(q_1,0,R)$	(q_2,Y,L)	-	(q_1,Y,R)	-
q_2	$(q_2,0,L)$	-	(q_0,X,R)	(q_2,Y,L)	-
q_3	-	-	-	(q_3,Y,R)	(q_4,B,R)
$*q_4$	-	--	-	-	-

Table representation of the state diagram



Turing Machine Problems and Solutions

Prashant Gautam

1. Turing Machine for L = $\{a^n\ b^n \mid n>=1\}$

- first all a's will come and then all b's will come.

  - **Input-1:aabb**
  - **Output-1:YES**

  - **Input-2:aabbbb**
  - **Output-2:NO**

  - **Input-3:abab**
  - **Output-3:NO**

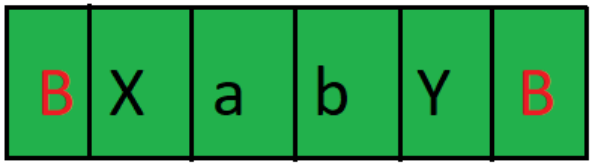
Approach

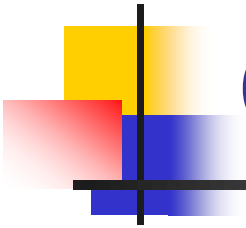
W= aabb

I. Scan the input from the left.

II. First, replace an ‘a’ with ‘X’ and move right. Then skip all the a’s and b’s and move right.

III. When the pointer reaches Blank(B) Blank will remain Blank(B) and the pointer turns left. Now it scans the input from the right and replaces the first ‘b’ with ‘Y’. Our Turing machine looks like this –





Contd…

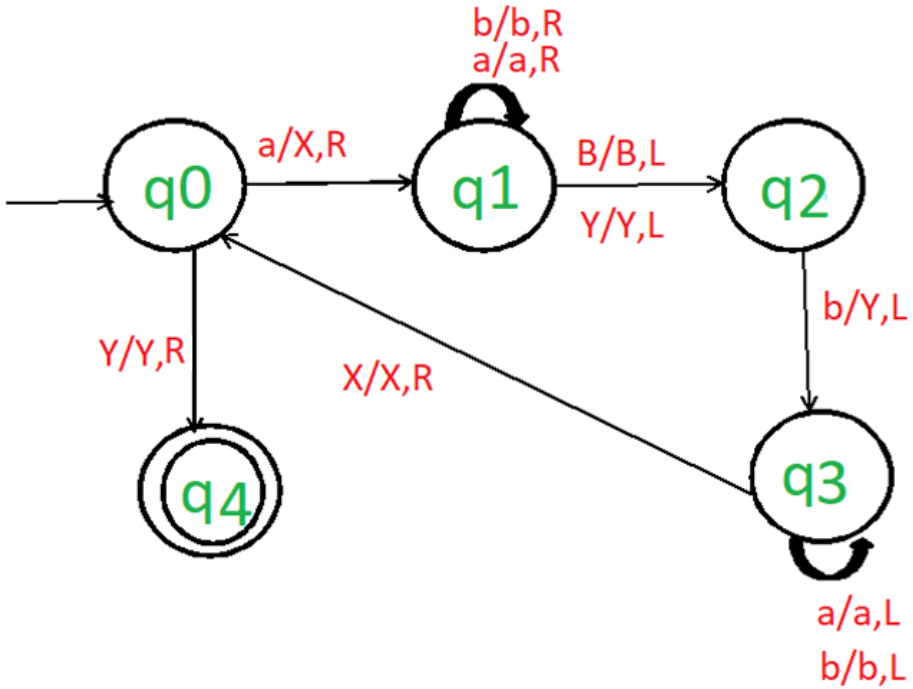
IV. Again the pointer reaches Blank(B) or X. It now scans the input from left to right. The pointer moves forward and replaces ‘a’ with ‘X’.

V. Again the pointer reaches Blank(B) or Y. It now scans the input from the right to left. The pointer moves forward and replaces ‘b’ with ‘y’.

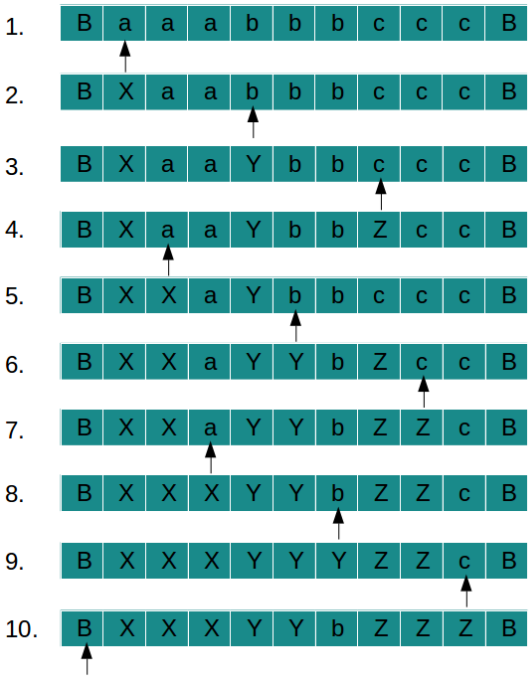
VI. We repeat the same steps until we convert all the a’s to ‘X’ and b’s to ‘Y’.

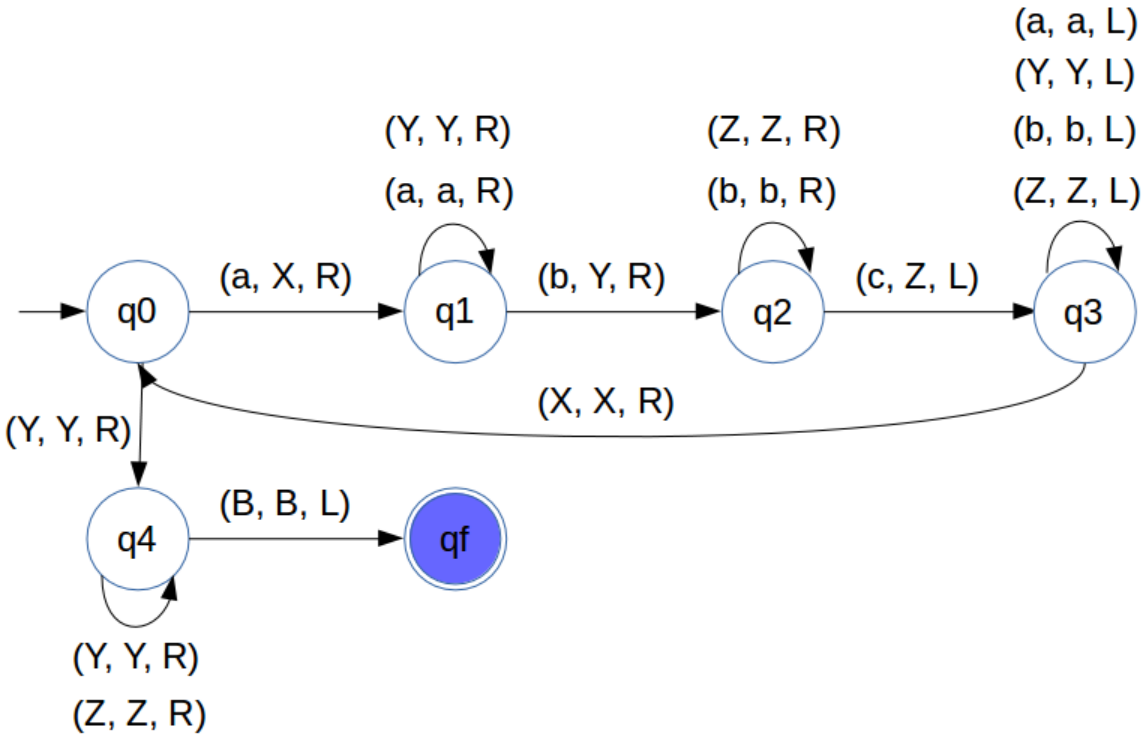
VII. When all the a’s converted to ‘X and all the b’s converted to ‘Y’ our machine will halt.

Solution



2. Turing Machine for L = $\{a^n b^n c^n \mid n>=1\}$

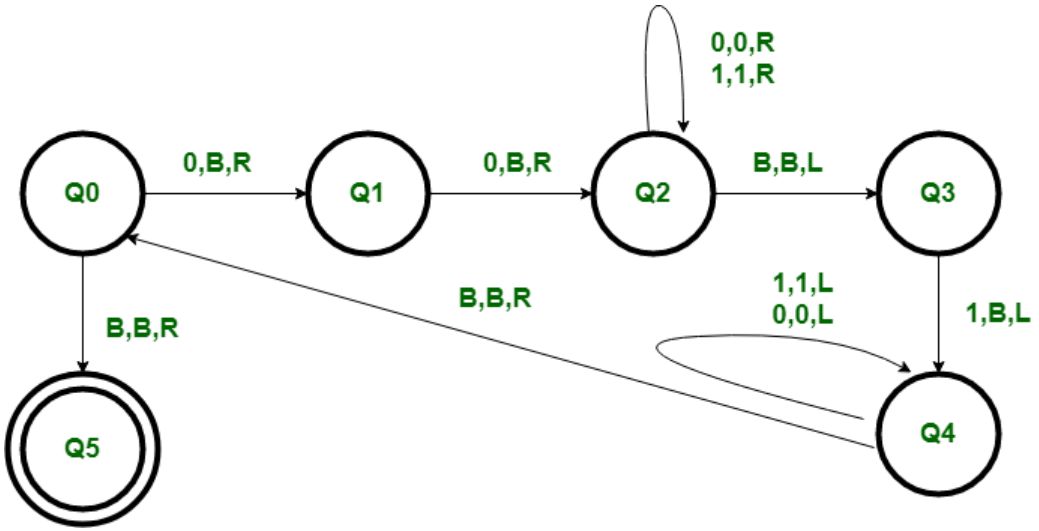




Construct a Turing Machine for language L = $\{0^{2n}1^{n} | n>=0\}$

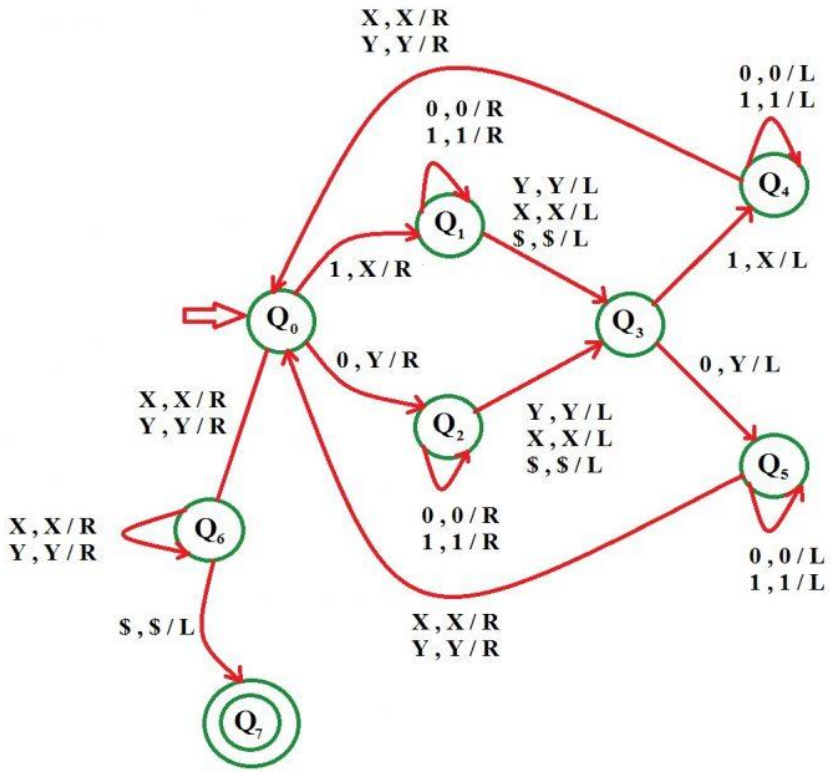
- Input : 001
- Output : YES

- Input : 00001
- Output : NO

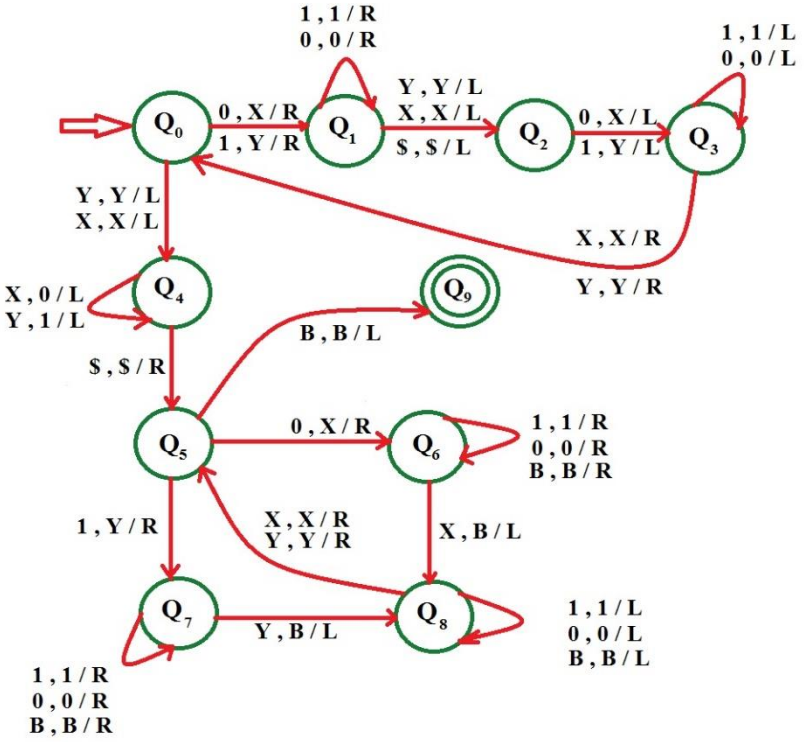


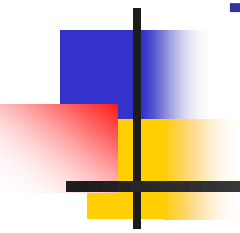
- Input : epsilon or empty string
- Output : YES

Construct a Turing Machine for language L = $\{WW^R | w \in \{0, 1\}\}$



Construct a Turing Machine for language L = {ww | w ∈ {0,1}}





Turing Machine Techniques

As a Language Recognizer

As a Function Computation

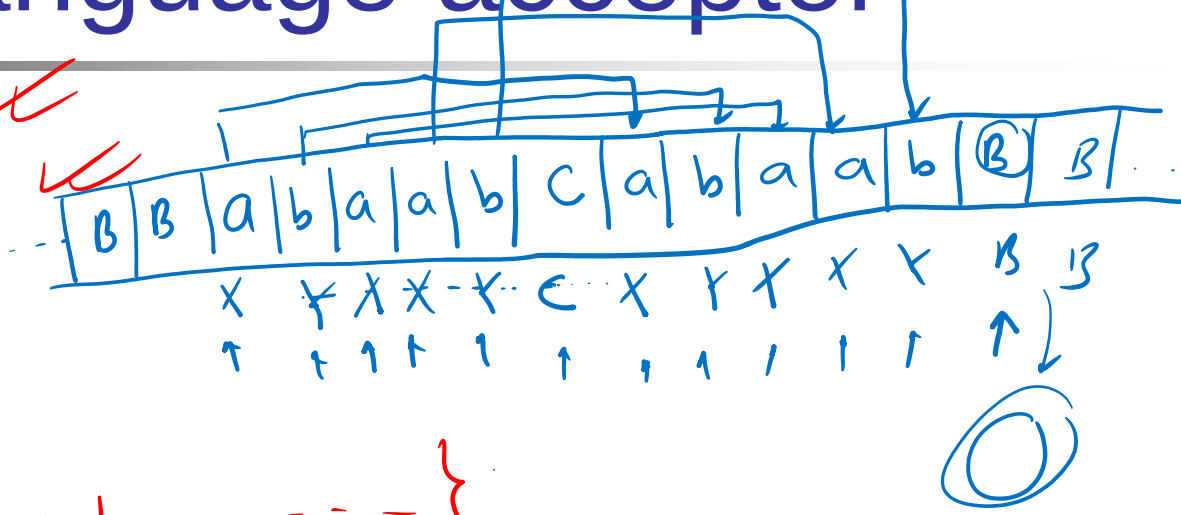
As a Transducer

TM as Language acceptor

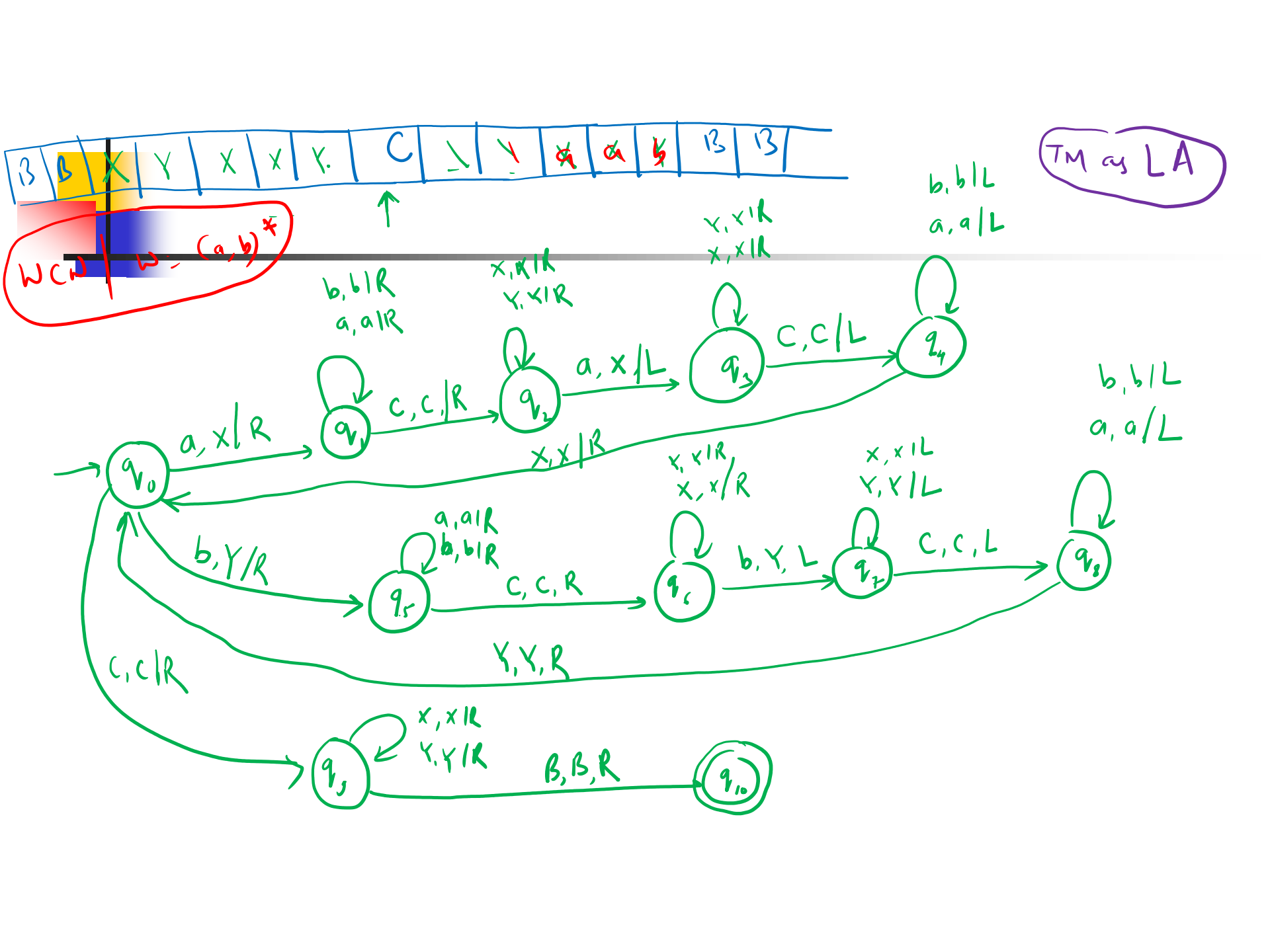
1) $a^n b^n \mid n >= 1$

2) $a^n b^n c^n \mid n >= 1$

3) $wcw \mid n >= 1$



$L = \{ \underbrace{abaab}_{w} \underbrace{c}_{c} \underbrace{abaab}_{w}, \ldots \}$


TM as LA
B B X Y X X Y C
WC
(a,b)*
b,b|L
a,a|L
X,X|R
X,X|R
b,b|R
a,a|R
X,X|R
Y,Y|R
C,C|L
a,X|L
C,C|R
a,X|R
X,X|R
b,b|L
a,a|L
X,X|R
X,X|L
Y,Y|L
a,a|R
b,b|R
b,Y|R
C,C,R
b,Y,L
C,C,L
C,C|R
Y,Y,R
X,X|R
Y,Y|R
B,B,R
q0
q1
q2
q3
q4
q5
q6
q7
q8
q9
q10

TM as Transducer

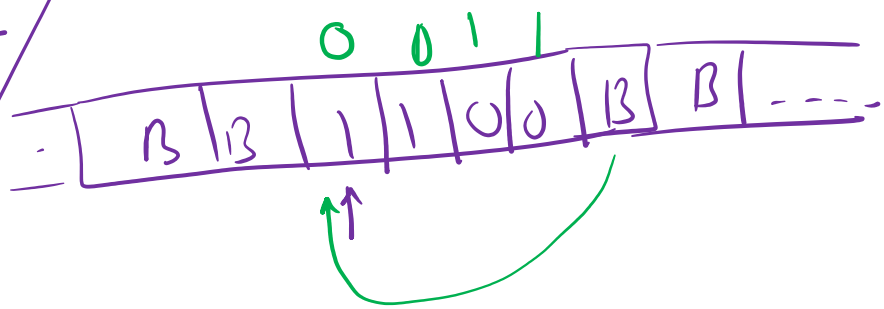
How? Logic.

Eg TM for 1's Complement

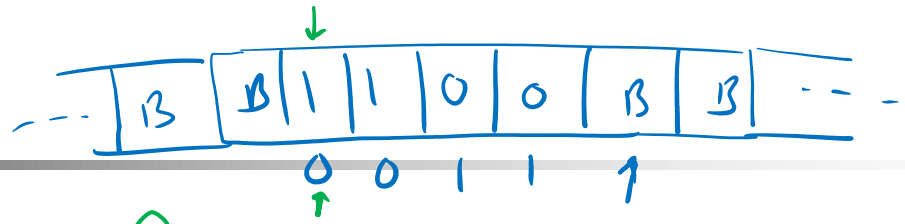
1 0 1 0 i/p

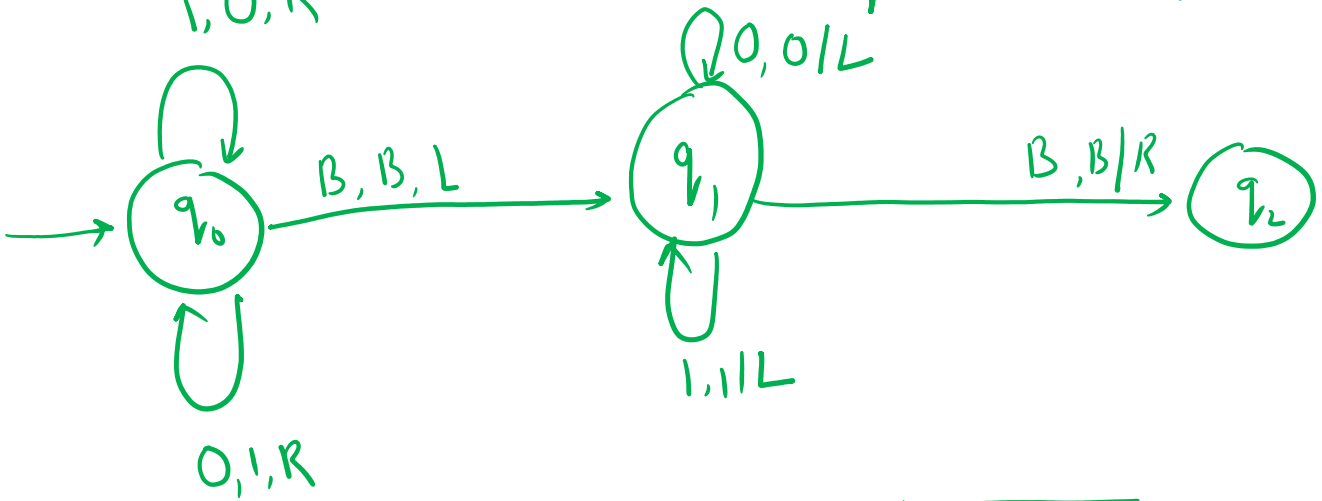
↓ ↓ ↓ ↓

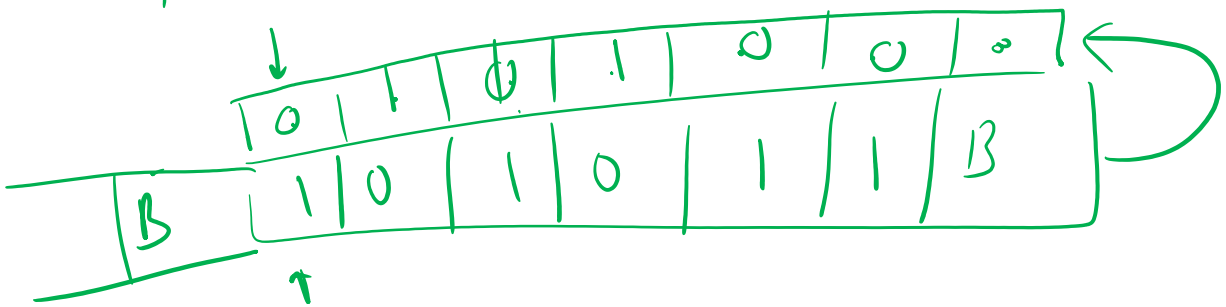
0 1 0 1 o/p



1's Complement.

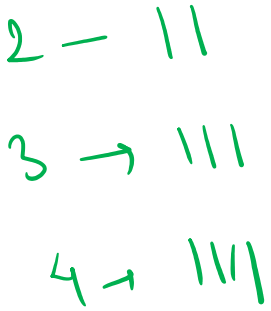


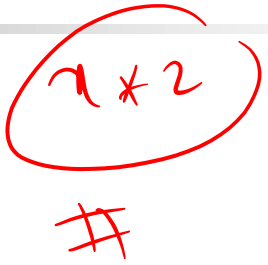




1's complement

TM as function computing



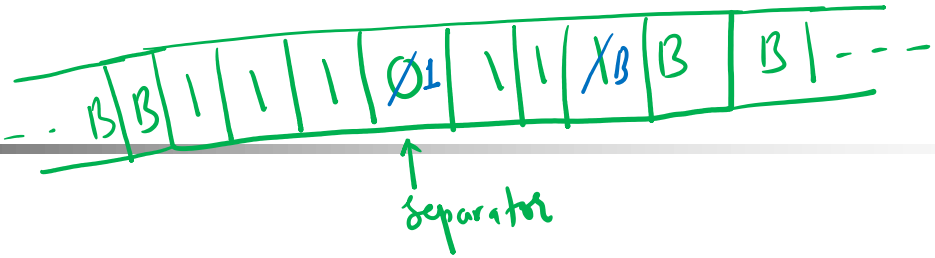


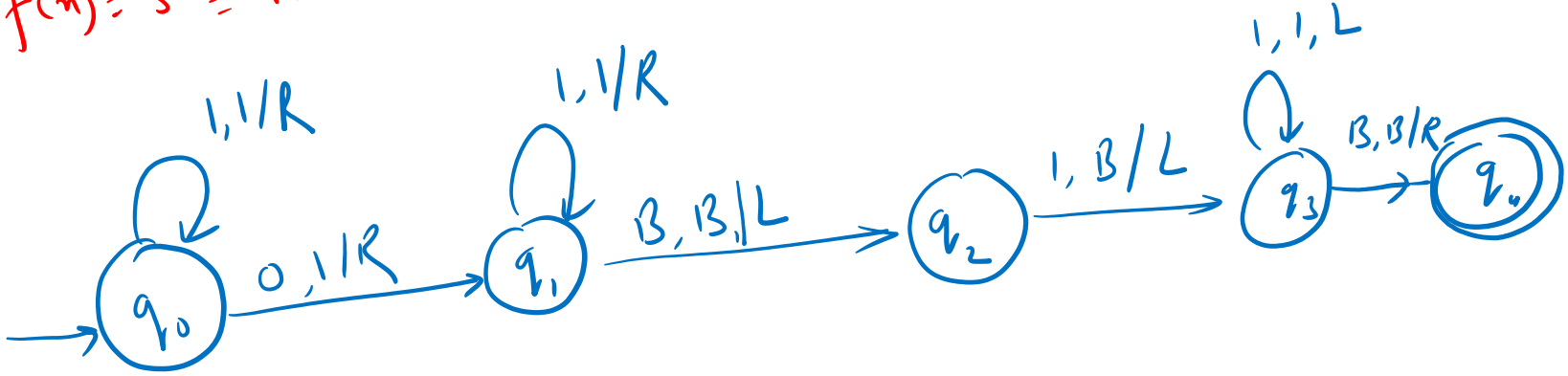
$f(n) = n + 2$

$n = 3 = 111$

$2 = 11$

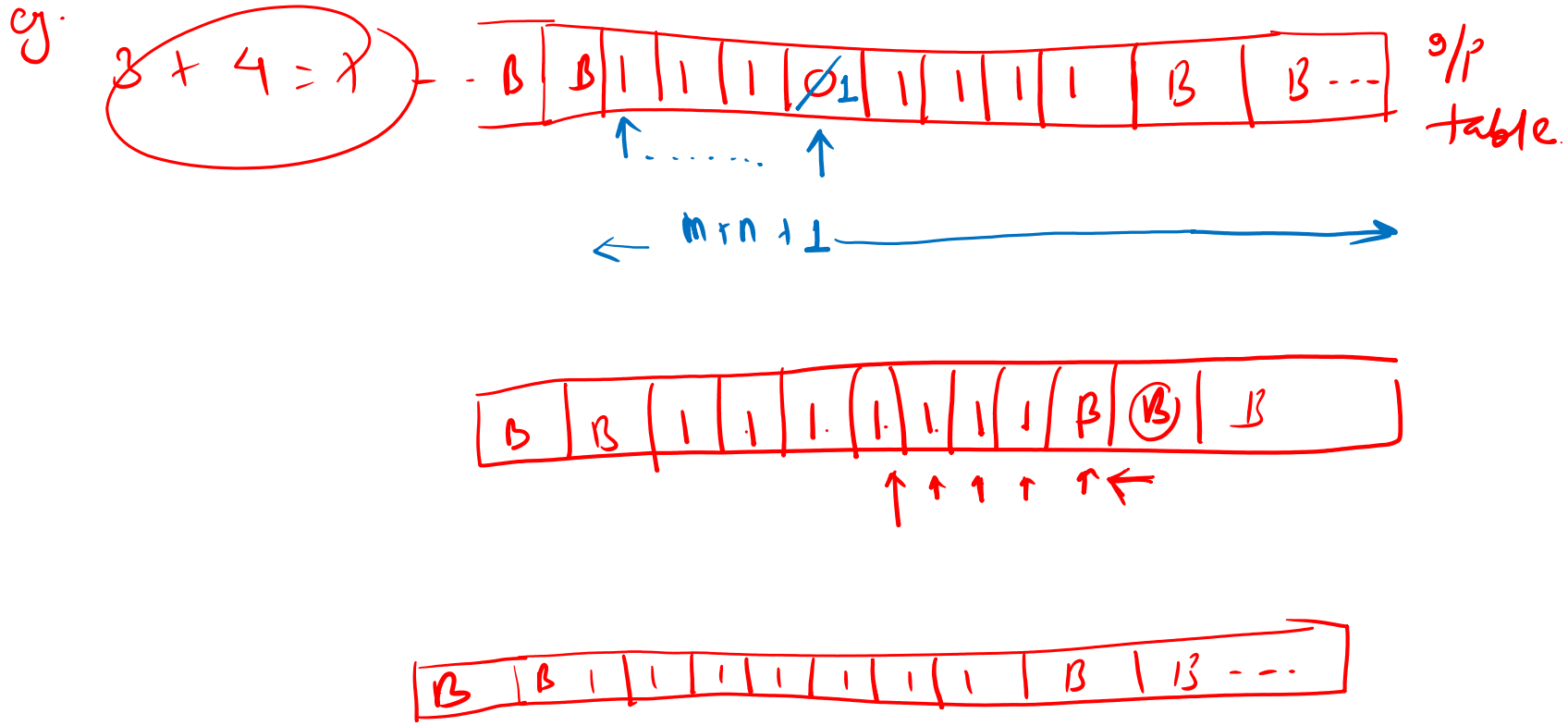
output $f(n) = 5 = 11111$

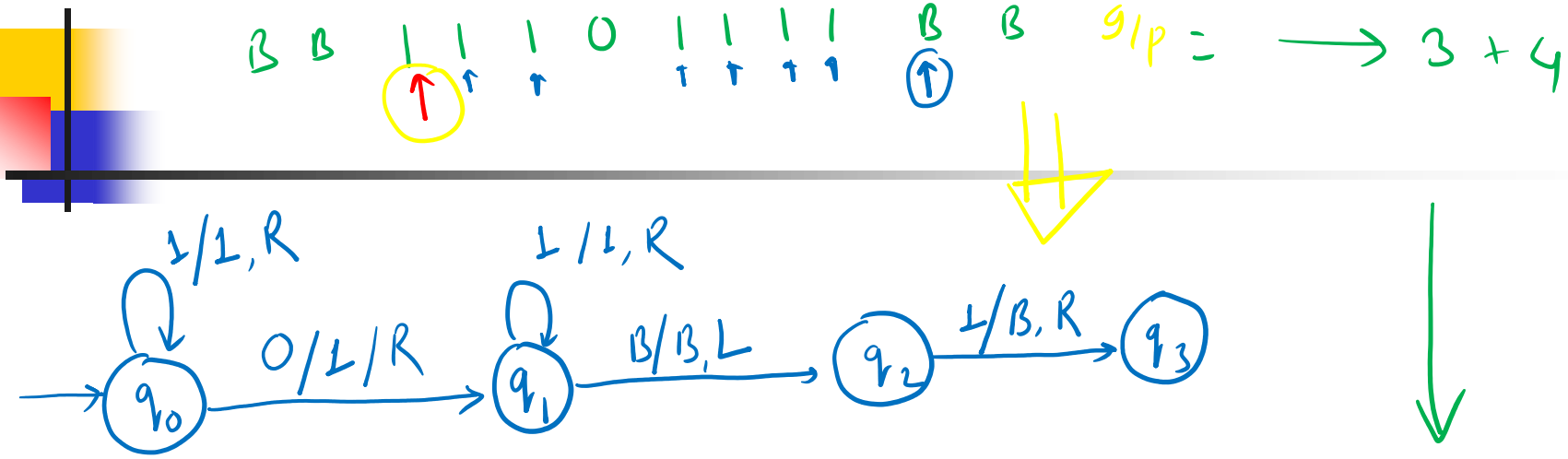




TM for function computing : Example

$m+n+1-1 = m+n$ (replacing B for 1).





TM as function Computation

B B 1 1 1 1 1 1 1 B B = 7

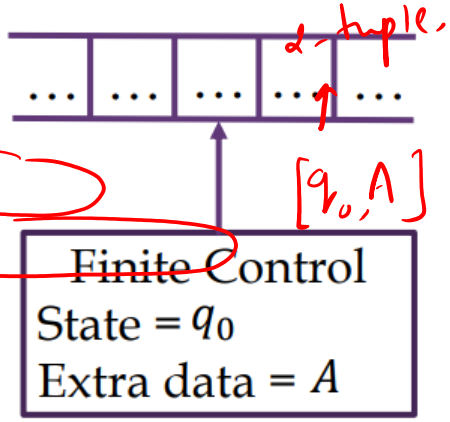
Turing Machine with Storage in its State

- We can keep track of a finite amount of extra data by incorporating it into the state of a TM.
- Example: Keep track of an additional symbol
- If the "extra data" can be A or B,

and the "state" can be $q0$ or $q1$,

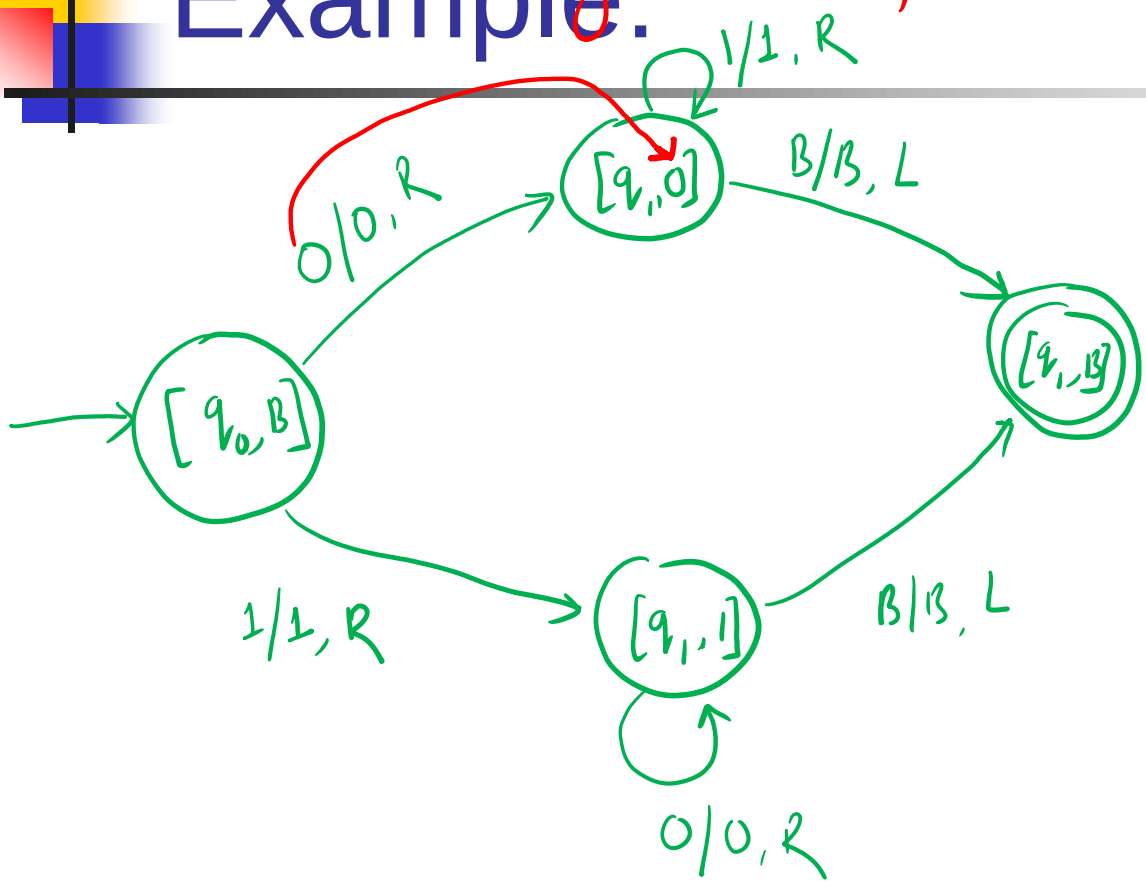
then the actual states of the TM can be

{[$q0$, A],[$q1$, A],[$q0$, B], [$q1$, B]}.

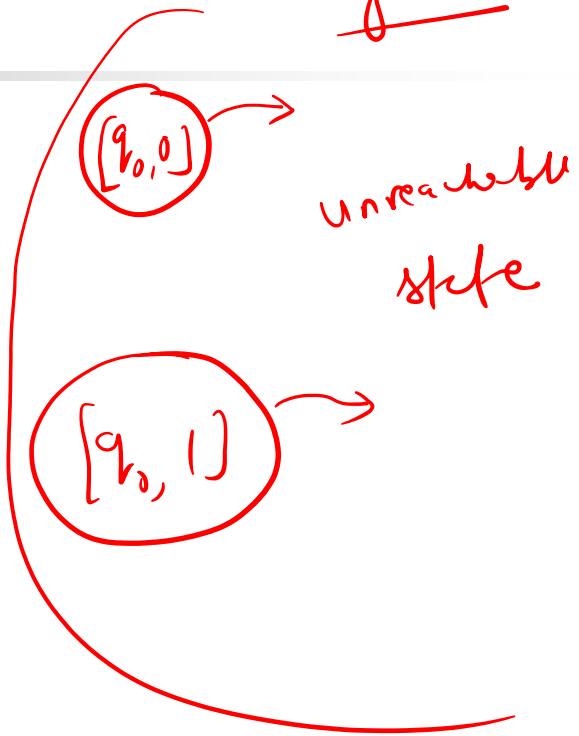


Example:

Q. Design a TM for $01^* + 10^*$

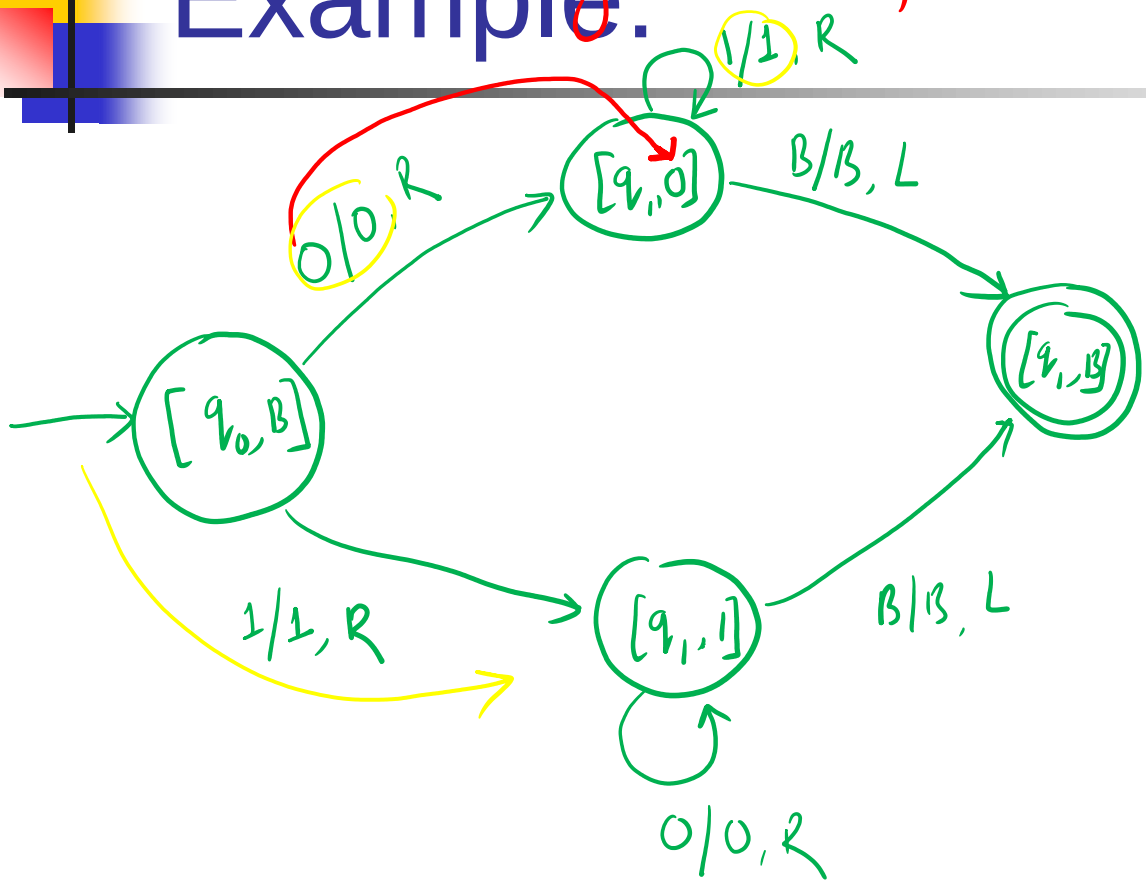


ignore

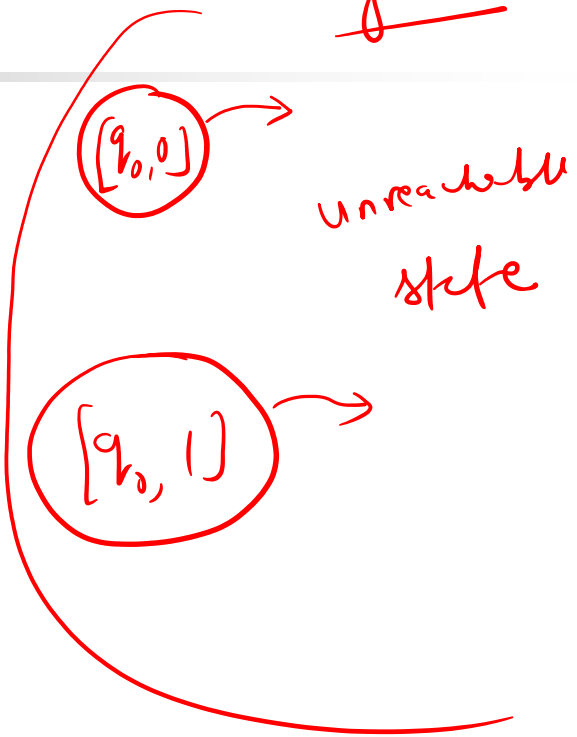


Example:

Q. Design a TM for $01^* + 10^*$



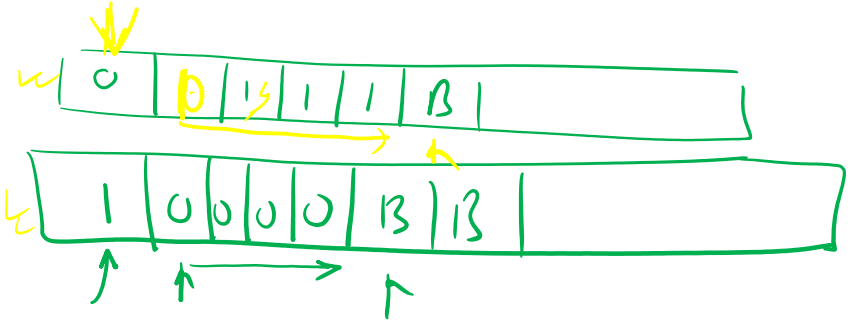
ignore



$\delta((q_0, B), 0) = ((q_1, 0), 0, R)$

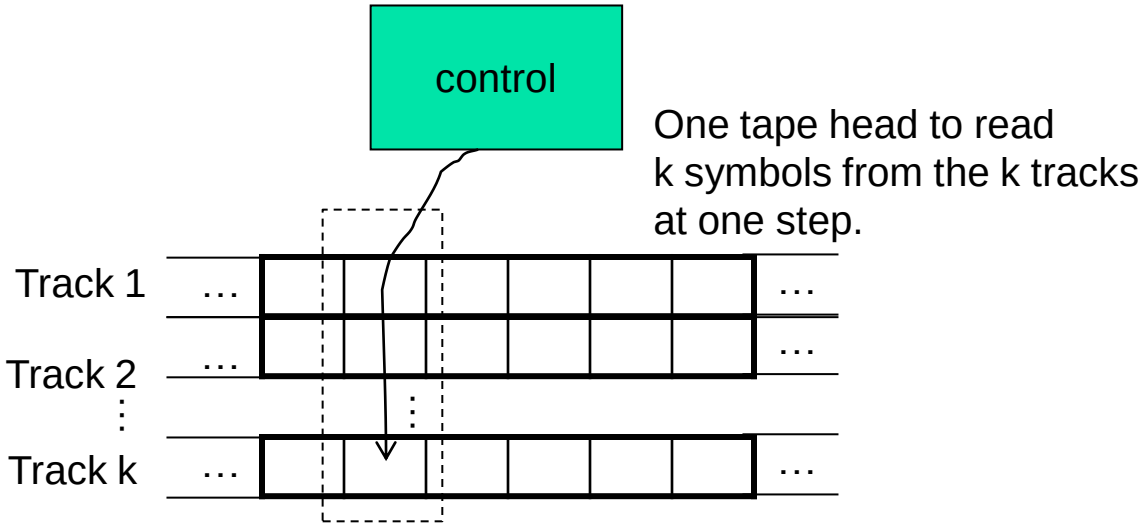
$\delta((q_1, 0), 1) = ((q_1, 0), 1, R)$

$\delta((q_1, 0), B) = ((q_1, B), B, L)$



Multi-track Turing Machines

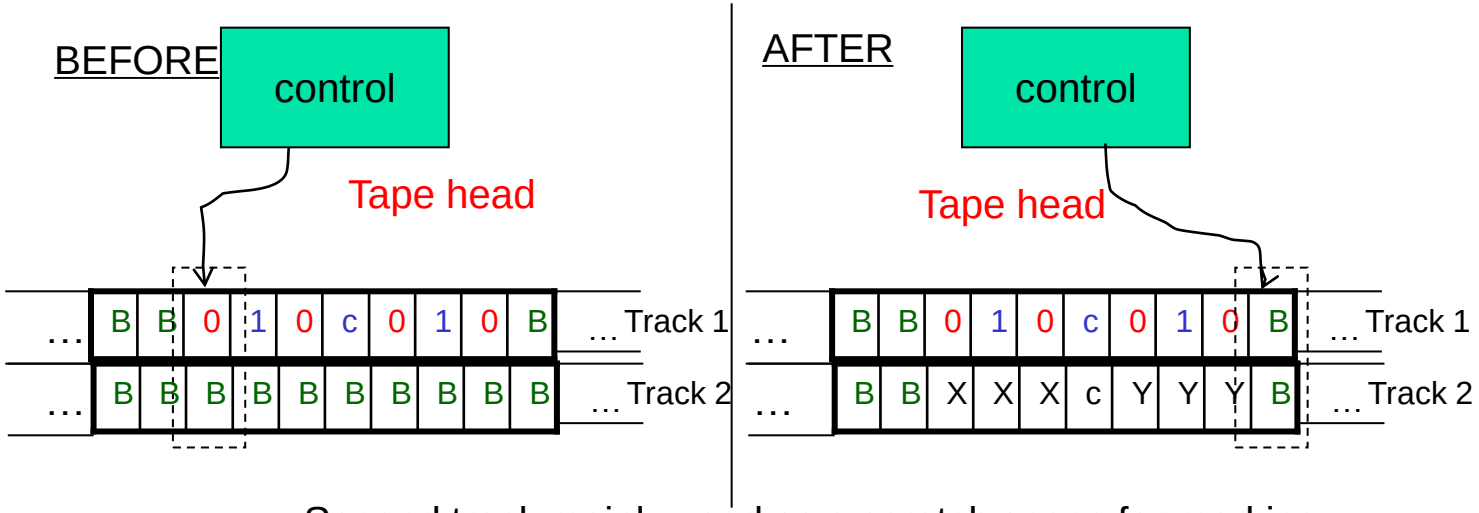
- TM with multiple tracks,
  but just one unified tape head



Multi-Track TMs

- TM with multiple “tracks” but just one head

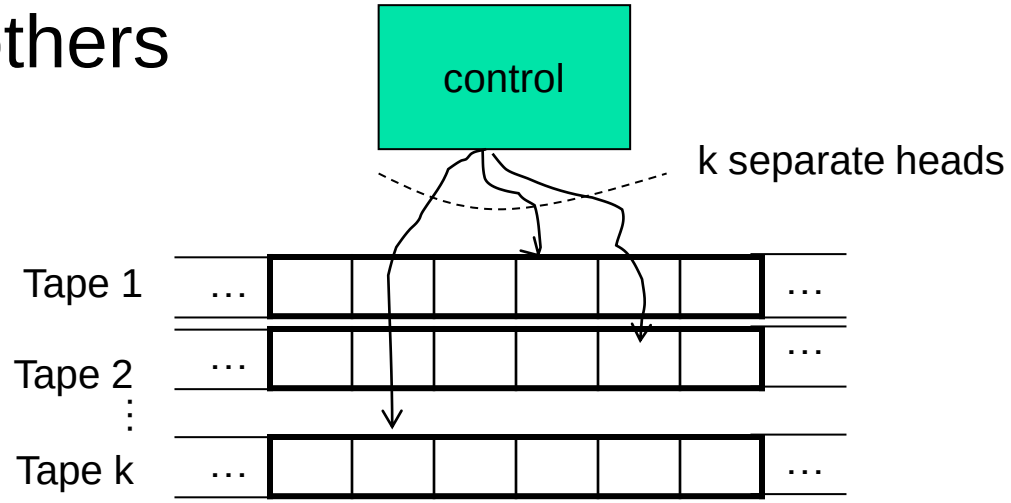
E.g., TM for {wcw | w∈ {0,1}* }
but w/o modifying original input string



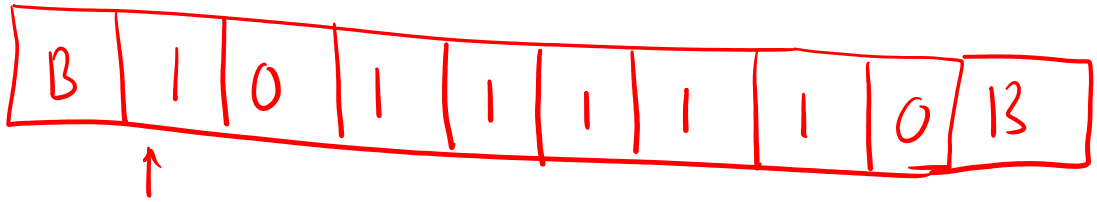
Second track mainly used as a scratch space for marking

Multi-tape Turing Machines

- TM with multiple tapes, *each tape with a separate head*
  - Each head can move independently of the others



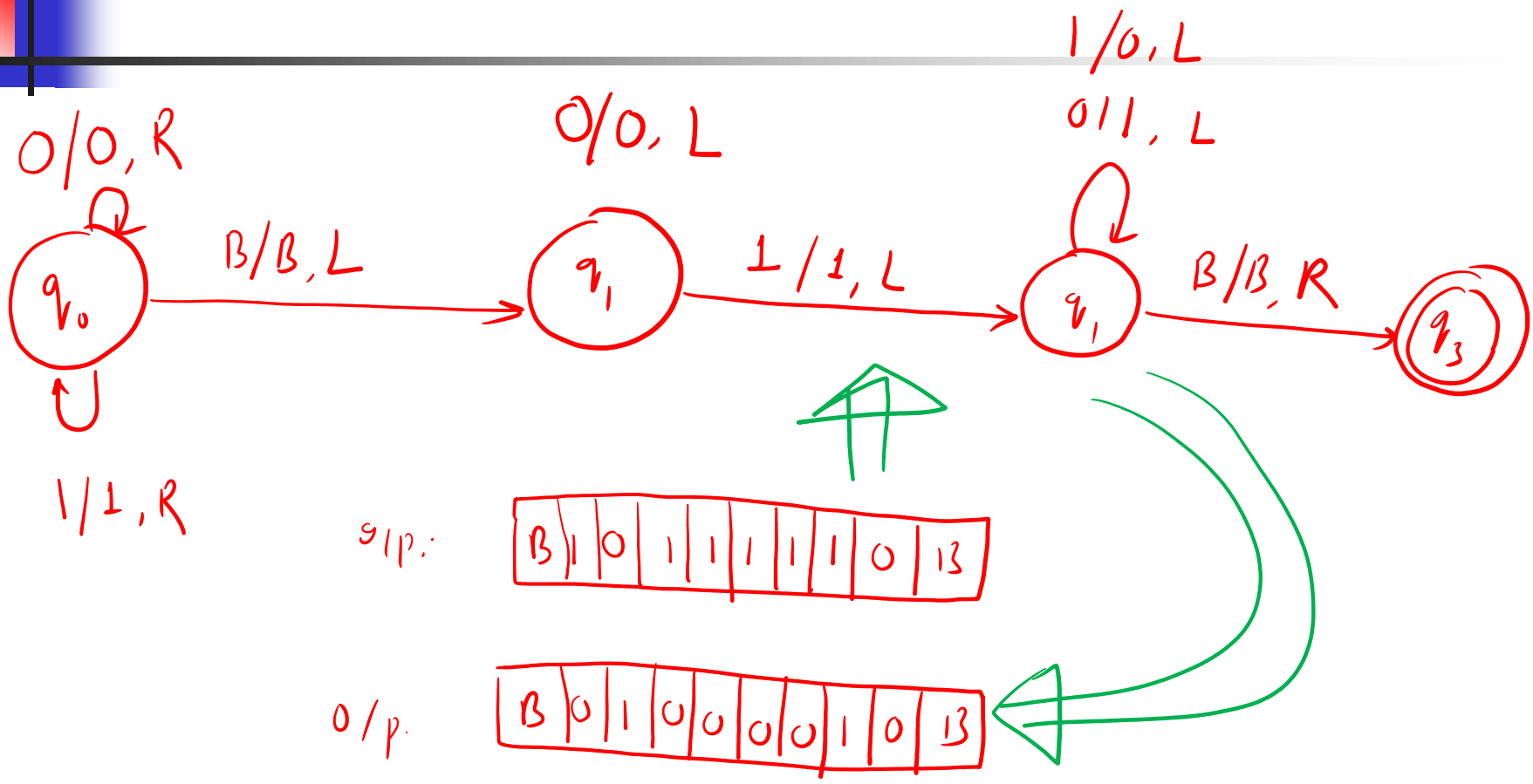
2s comp of a binary string using TM



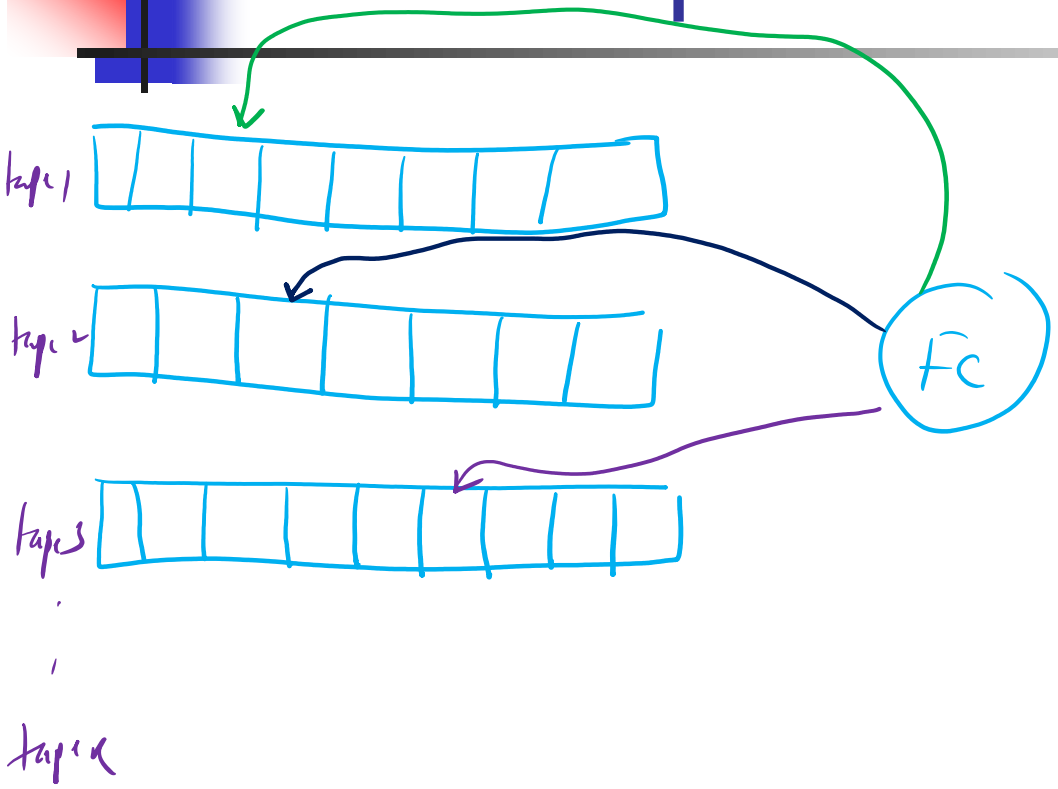
1010
1
1011

Logic:

1. Scan R → L
2. passing all cons. 0's
3. 1 → 1/1 ←
4. all one 0 all 0 1.


0/0, R
0/0, L
1/0, L
0/1, L
B/B, L
1/1, L
B/B, R
1/1, R
q0
q1
q1
q3
i/p:
B 1 0 1 1 1 1 1 0 B
o/p:
B 0 1 0 0 0 0 1 0 B

Multi-Tape TMs



Equivalence of one-tape and Multi-tape TM

- Any recursively enumerable languages that are accepted by one-tape TM are also accepted by multi-tape TM. i.e. Any n-tuples TM for n ≥ 2, are at least as powerful as 1-tape TM's.

1. Same power
2. Accepts same set of Lang
3. Differ in Computational easeness
4. Differ in time.

Simulating one tape TM with multi-tape TM

Theorem

- Every language accepted by a Multi-tape TM is recursively enumerable.

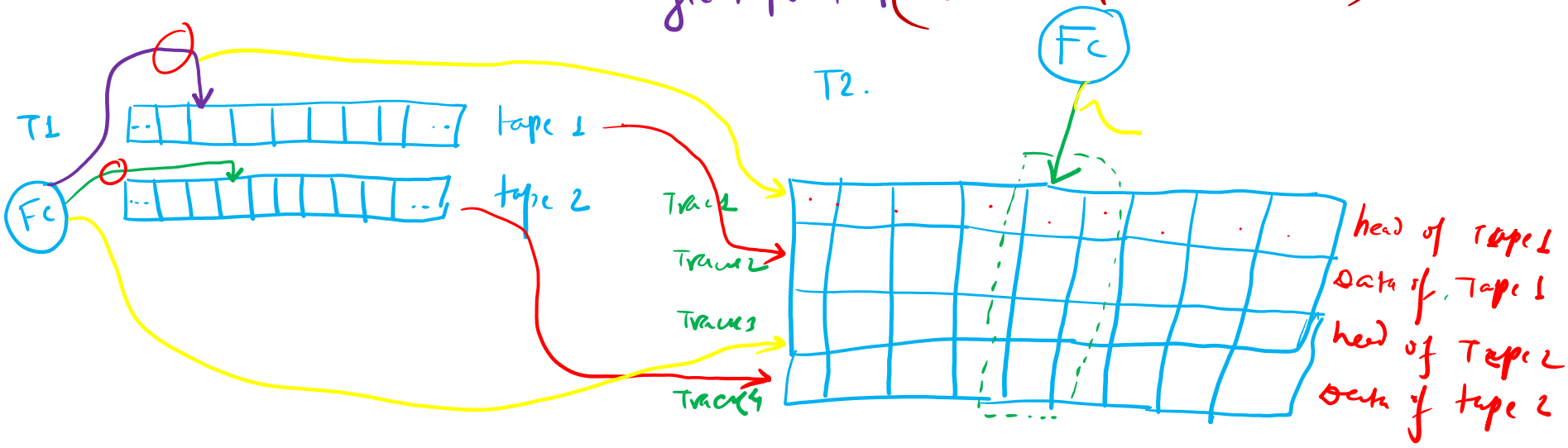
Or,

- Any languages that are accepted by a multi-tape TM are also accepted by one tape Turing Machine.

Proof:

Suppose: T1 → Multitape TM (n = 2)

T2 → single tape TM (2n tracks ie. 4 tracks)



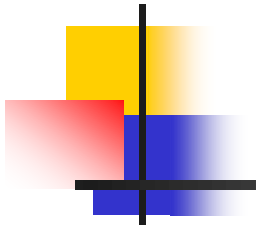
Non-Deterministic Turing Machine

$\delta(q,x) = (q_1, Y_1, D_1), (q_2, Y_2, D_2), (q_3, Y_3, D_3)$

- A non-deterministic Turing Machine (NTM), T = (Q, Σ, Γ, δ, q0, B, F) is defined exactly the same as an ordinary TM, except the value of transition function δ.
- In NTM, the values of the transition function δ are subsets, rather than a single element of the set Q×Γ×{R,L,S}.
- Here, the transition function δ is such that for each state q and tape
- symbol x, δ(q, x) is a set of triples.

{(q1, Y1, D1), (q2, Y2, D2), ...(qk, Yk, Dk) where k is

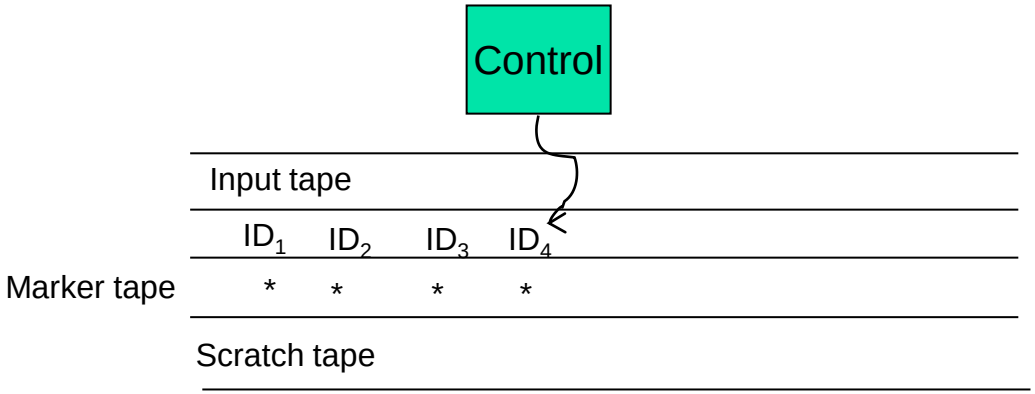
any finite integer.



- The NTM can choose, at each step, any of the triples to be the next move. It cannot, however pick a state from one, a tape symbol from another, and the direction from yet another.

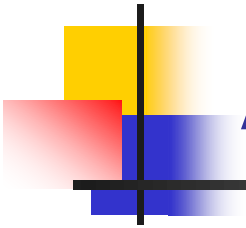
Non-deterministic TMs

- A TM can have non-deterministic moves:
  - $\delta(q,X) = \{ (q_1,Y_1,D_1), (q_2,Y_2,D_2), \ldots \}$
- Simulation using a multitape deterministic TM:



Point to be noted

- ***Basic TM is <u>equivalent</u> to all the below:***
  1. *TM + storage*
  2. *Multi-track TM*
  3. *Multi-tape TM*
  4. *Non-deterministic TM*



Assignment

- ***Prepare a note on following Topics***

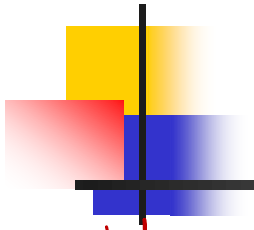
Restricted Turing Machines:

1) With Semi-infinite Tape,
2) Multi-stack Machines,
3) Counter Machines

Church-Turing Thesis

Alonzo Church (1903-1995)

Alan Turing (1912-1954)



Statement:

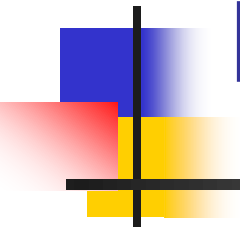
"A functional on natural number is computable by an algorithm if and only if it is computable by turing Machine."

Supporting stmts:

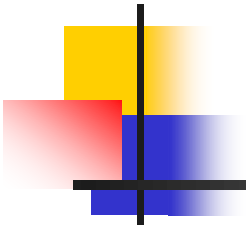
① anything that can be done by current digital computer can be done by a TM.

② currently there is no problem which can be solved by a digital computer and cannot be solved by a TM.

③ Many mathematical model are suggested but no one of them is more powerful than TM.



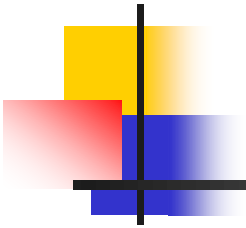
Decidability and Undecidability



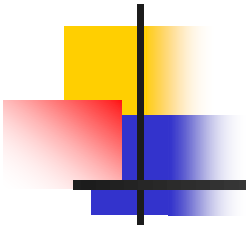
Recursive language(REC)

- A language ‘L’ is said to be recursive if there exists a Turing machine which will accept all the strings in ‘L’ and reject all the strings not in ‘L’.
- The Turing machine will halt every time and give an answer(accepted or rejected) for each and every string input.
- A language ‘L’ is decidable if it is a recursive language.
- All decidable languages are recursive languages and vice-versa.

Recursively enumerable language(RE)



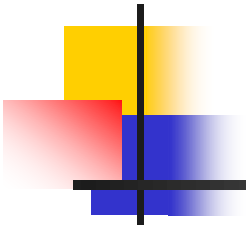
- A language ‘L’ is said to be a recursively enumerable language if there exists a Turing machine which will accept (and therefore halt) for all the input strings which are in ‘L’ but may or may not halt for all input strings which are not in ‘L’.
- By definition , all REC languages are also RE languages but not all RE languages are REC languages.



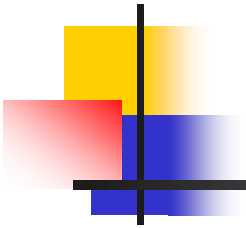
Decidable language

- A decision problem P is said to be decidable (i.e., have an algorithm) if the language L of all yes instances to P is decidable.
- Given a TM does it accept a given word?
- Recursive Langauge

Partially decidable or Semi-Decidable Language



- A language ‘L’ is partially decidable if ‘L’ is a RE but not REC language.
- **Recursively enumerable language(RE)**



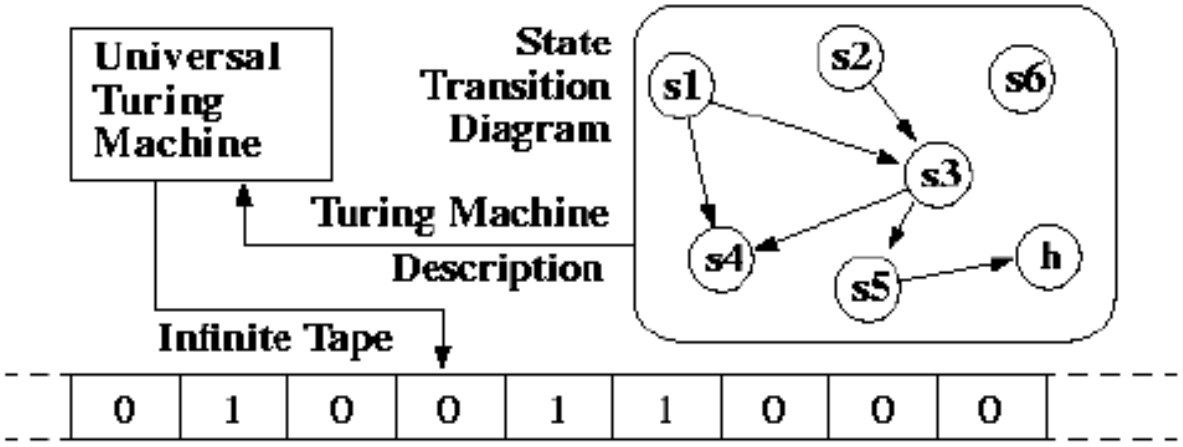
Undecidable language

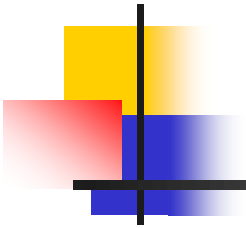
- An undecidable language maybe a partially decidable language or something else but not decidable.
- If a language is not even partially decidable , then there exists no Turing machine for that language.

Decidability & Undecidablity

Recursive Language	TM will always Halt
Recursively Enumerable Language	TM will halt sometimes & may not halt sometimes
Decidable Language	Recursive Language
Partially Decidable Language	Recursively Enumerable Language
UNDECIDABLE	No TM for that language

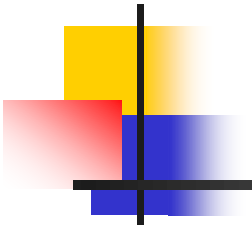
Universal Turing Machine





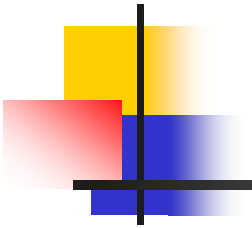
Universal Turing Machine

- A Turing Machine is the mathematical tool equivalent to a digital computer.
- The model consists of an input output relation that the machine computes.
- (UTM), which along with the input on the tape, takes in the description of a machine M.
- The UTM can go on then to simulate M on the rest of the contents of the input tape. A universal Turing machine can thus simulate any other machine.



- We shall construct a Turing Machine Mu, that takes as input a description of a Turing Machine M and an input word x, and simulates the computation of M on input x.
- A machine such as Mu that can simulate the behavior of an arbitrary TM is called a Universal Turing Machine.
- Thus, we can describe a Universal Turing Machine Tu as a TM, that on input <M, w>;

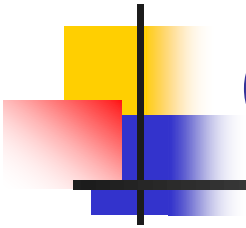
where M is a TM and w is string of input alphabets, simulates the computation of M on input w.



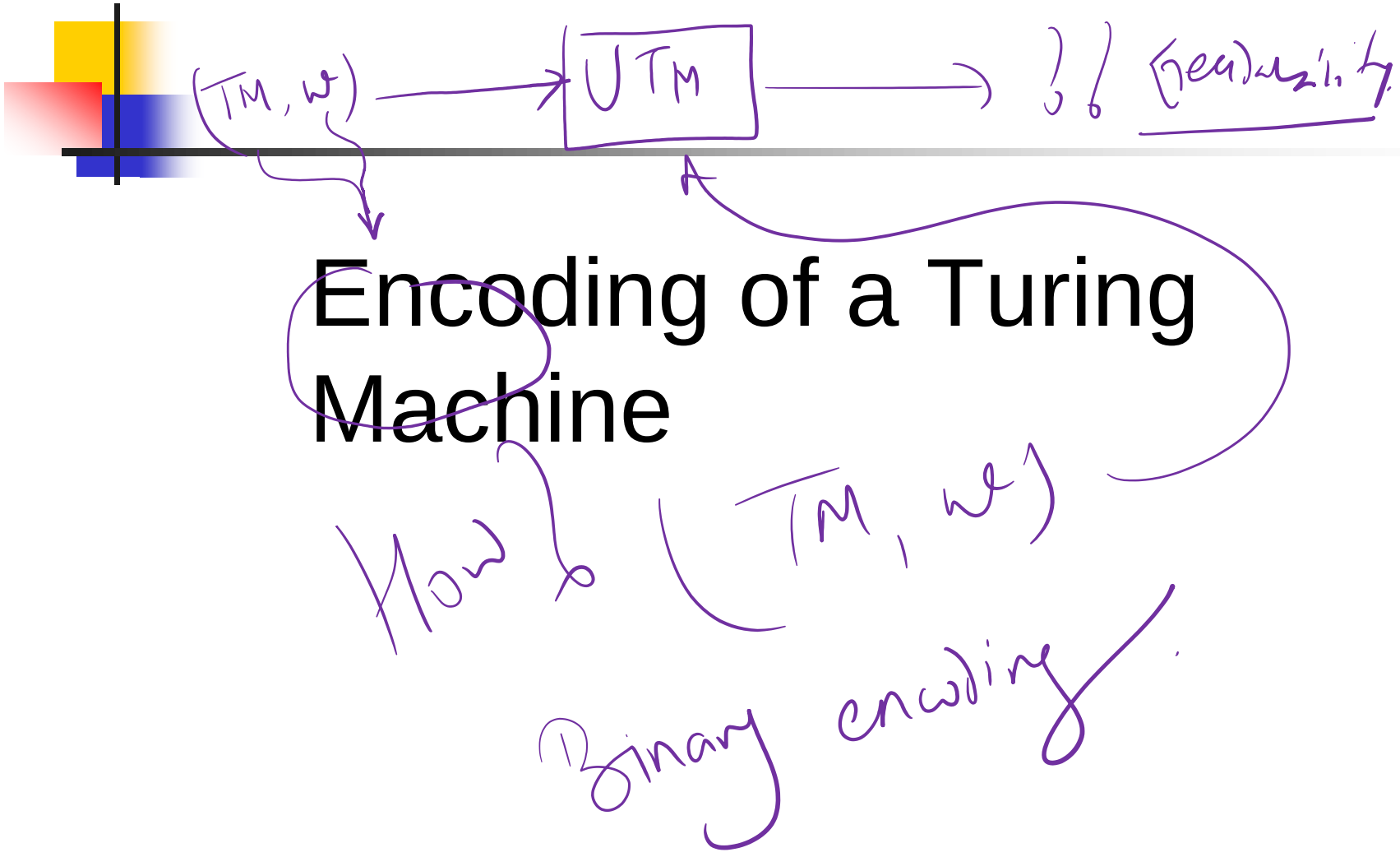
- specially,

- Tu accepts <M, w> iff M accepts w.
- Tu rejects <M, w> iff M rejects w.

Turing Machine and Computers



- Similarity
- Difference



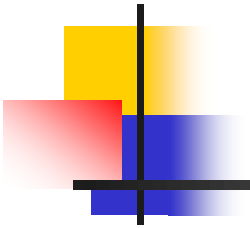
Encoding of a Turing Machine

A limitation of Turing Machines:

Turing Machines are "hardwired"

they execute only one program

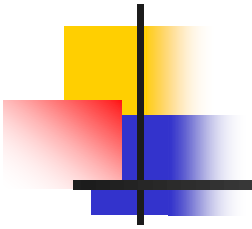
Real Computers are re-programmable



Solution: Universal Turing Machine

Attributes:

- Reprogrammable machine
- Simulates any other Turing Machine



Universal Turing Machine

simulates any Turing Machine M

Input of Universal Turing Machine:

Description of transitions of M

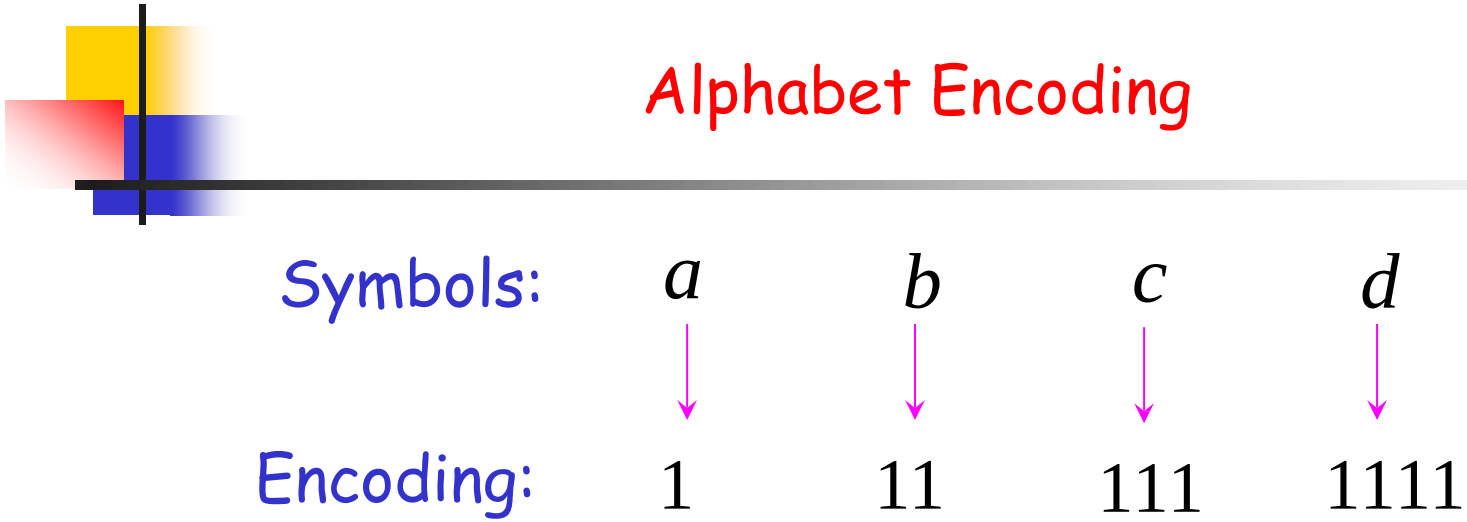
Input string of M

Description of M

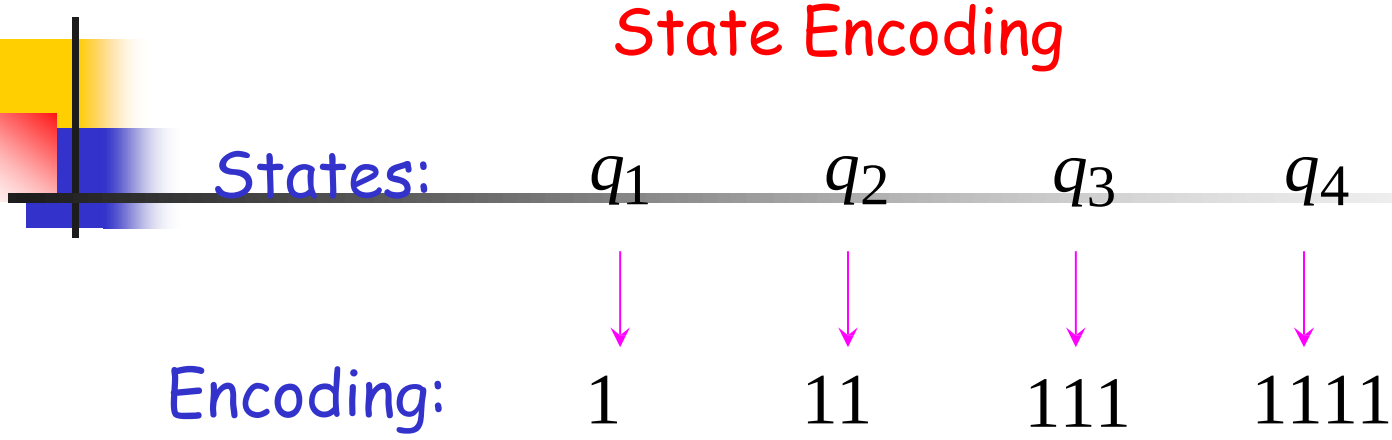
We describe Turing machine M
as a string of symbols:

We encode M as a string of symbols

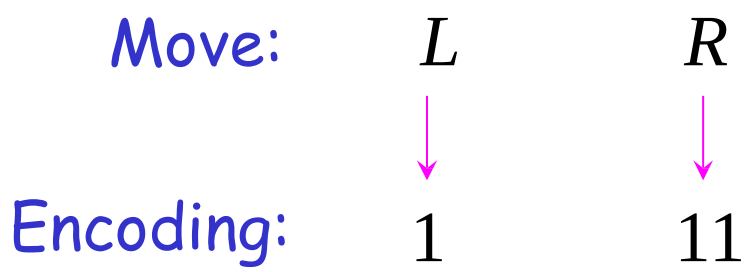
Alphabet Encoding

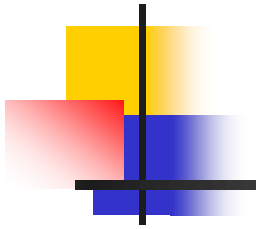


State Encoding



Head Move Encoding





Transition Encoding

Transition: $\delta(q_1, a) = (q_2, b, L)$

Encoding: $1\,0\,1\,0\,1\,1\,0\,1\,1\,0\,1$

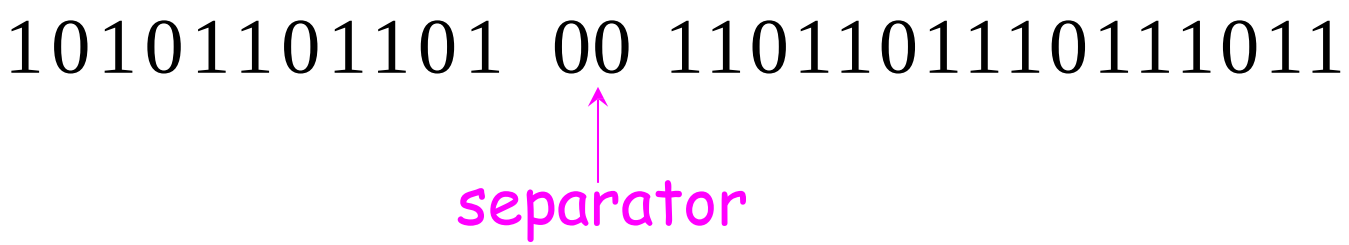
separator

Turing Machine Encoding

Transitions:

$$\delta(q_1, a) = (q_2, b, L) \qquad \delta(q_2, b) = (q_3, c, R)$$

Encoding:

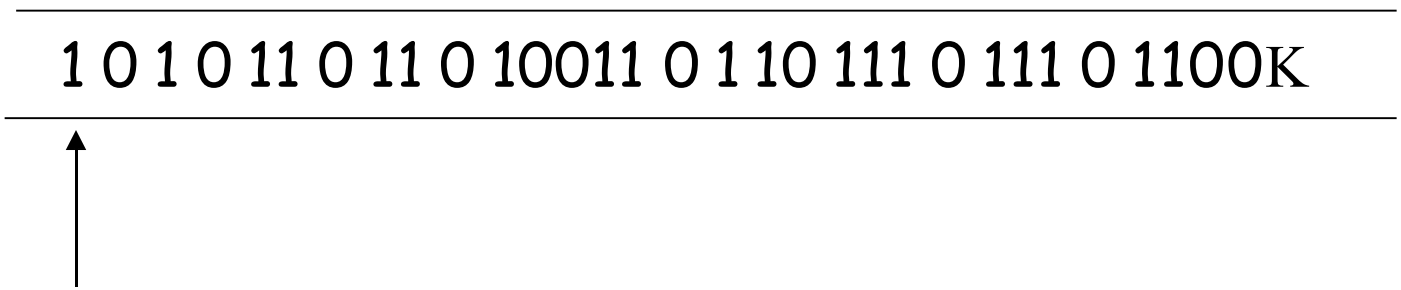


Tape 1 contents of Universal Turing Machine:

binary encoding

of the simulated machine M

Tape 1



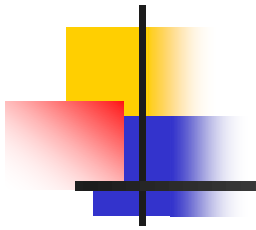
A Turing Machine is described with a binary string of 0's and 1's

Therefore:

The set of Turing machines
forms a language:

each string of this language is
the binary encoding of a Turing Machine

Language of Turing Machines

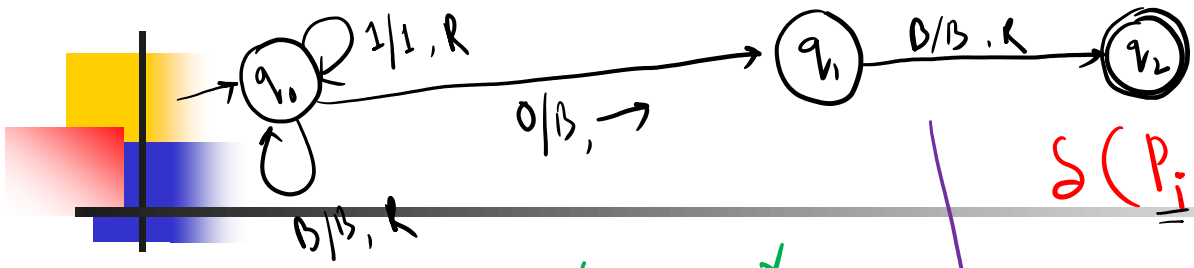


L = { 010100101, (Turing Machine 1)

00100100101111, (Turing Machine 2)

1110100111110010101, ...

..... }



1) Symbol : $X_1, X_2, \cdots X_3$

2) States : $P_1, P_2, P_3 \cdots$

3) Direction : D_1, D_2

$q_1 \rightarrow P_1$

$q_2 \rightarrow P_2$

$q_3 \rightarrow P_3$

$0 \rightarrow X_1$

$1 \rightarrow X_2$

$B \rightarrow X_3$

$R \rightarrow D_1$

$L \rightarrow D_2$

Formula

$\delta(P_i, X_j) = (P_k, X_l, D_m)$

Encode

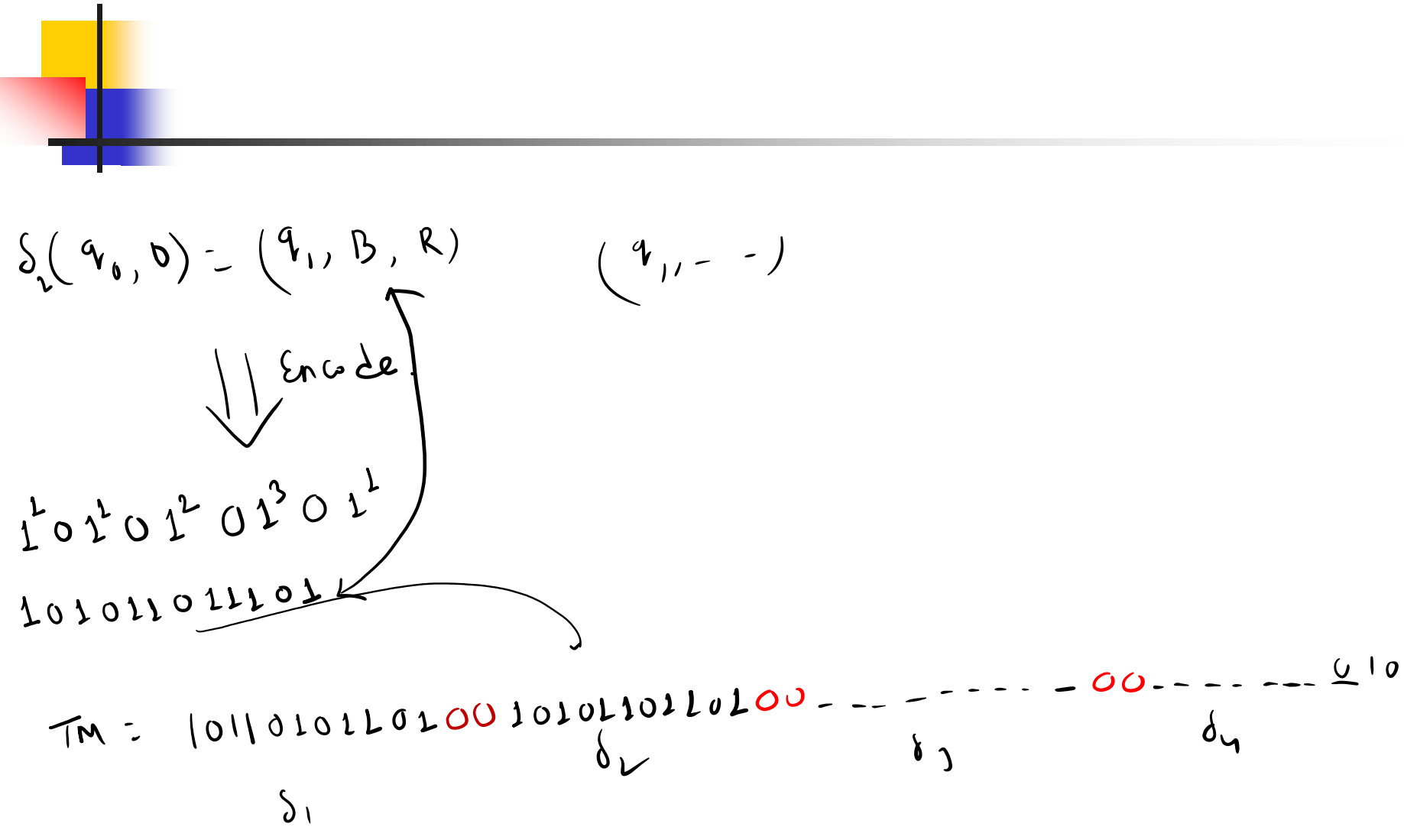
$1^i 0 1^j 0 1^k 0 1^l 0 1^m$

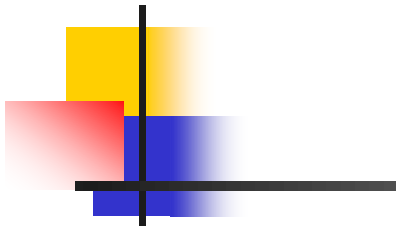
$\delta_1(q_0, 1) = (q_0, 1, R)$

Encode

$1^1 0 1^2 0 1^1 0 1^2 0 1^1$

10110101101


$\delta_2(q_0, 0) = (q_1, B, R)$
$(q_1, -, -)$
Encode
$1^1 0 1^1 0 1^2 0 1^3 0 1^1$
1010110111101
TM : 1011010110100101011010110100 ---------- 00 ---------- 0 10
δ_2
δ_3
δ_4
δ_1



$\delta(q_1, B) = (q_2, B, R)$

1) Rewriting this as per our prev. Convension

$\delta(P_2, X_3) = (P_3, X_3, D_2)$

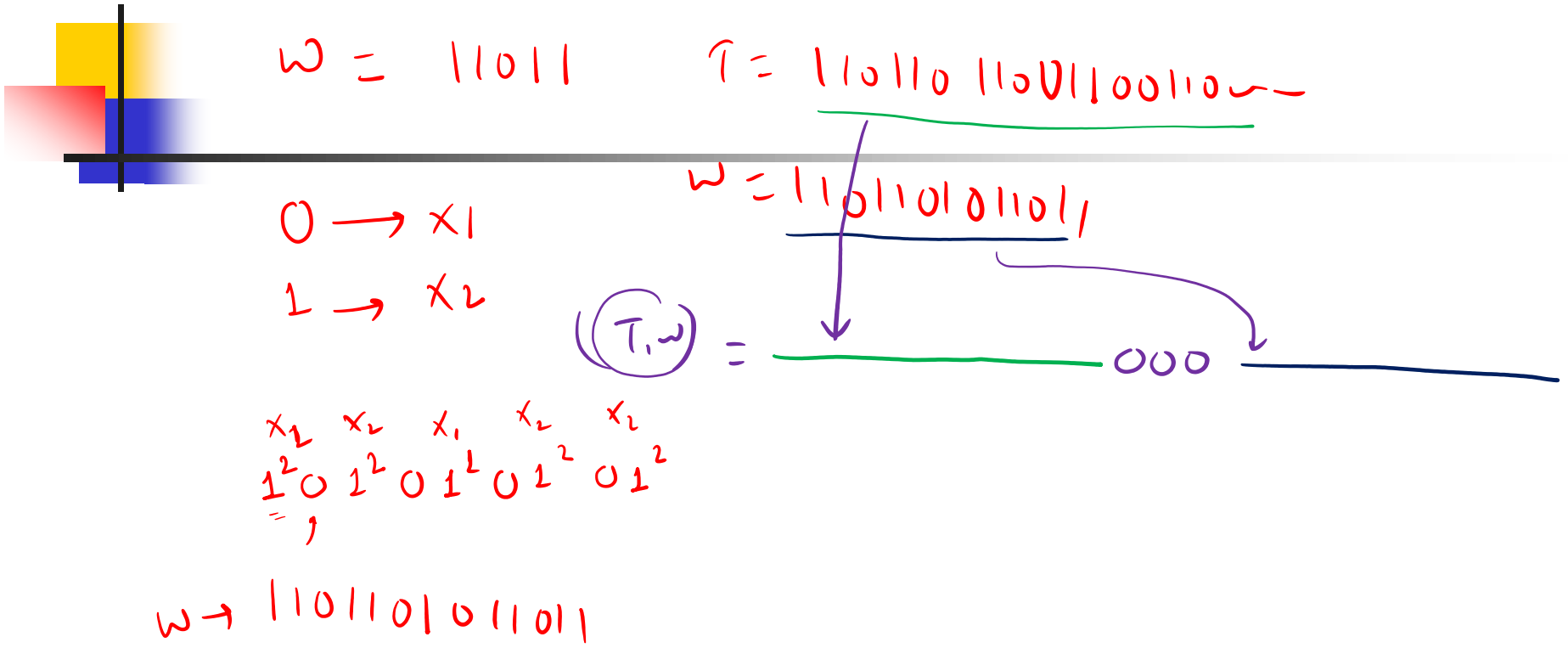
$1^2 0 1^3 0 1^3 0 1^3 0 1^2$

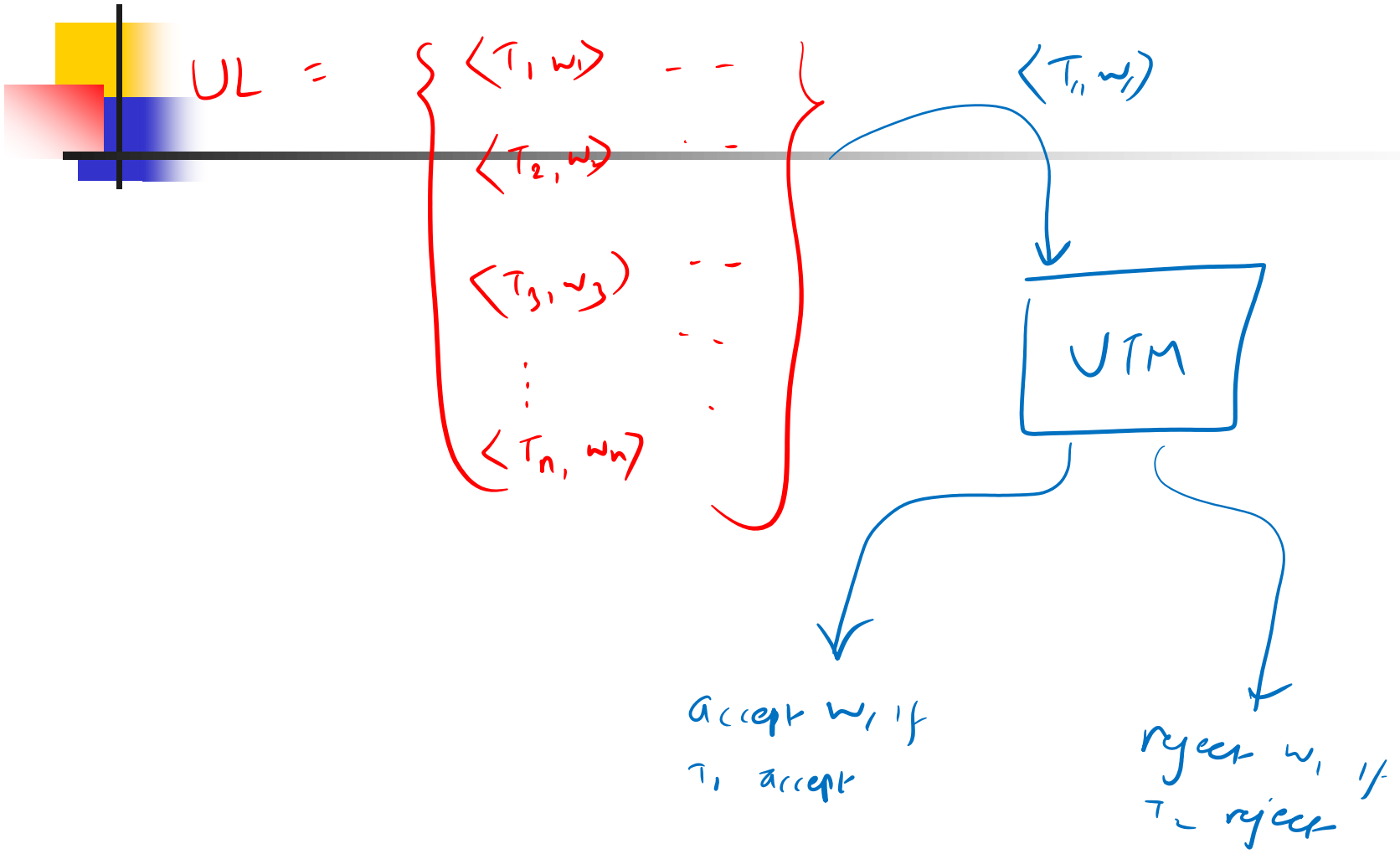
1101110111011101

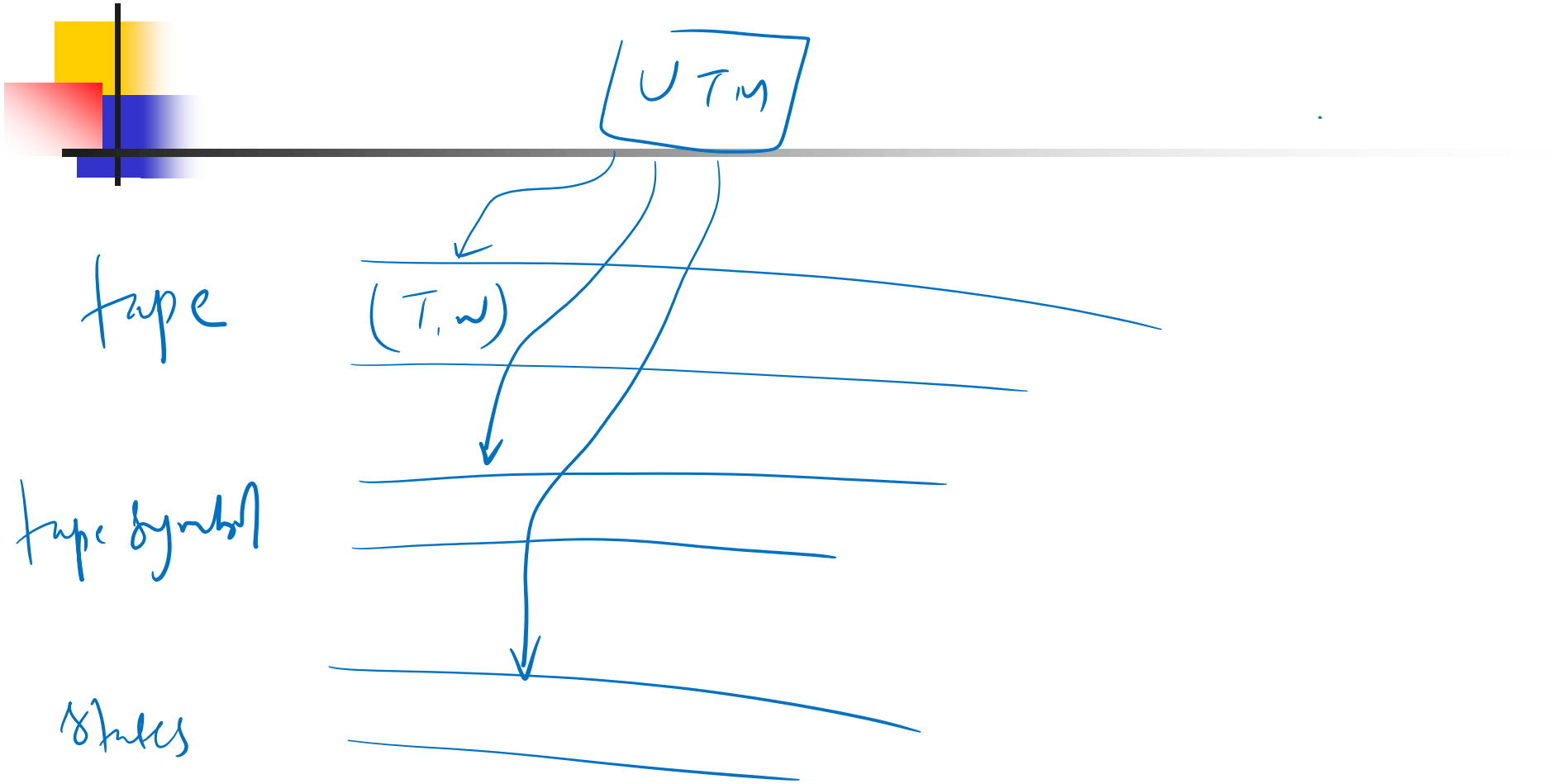
formula:

$\delta(P_i, X_j) = (P_k, X_l, D_m)$

$1^i 0 1^j 0 1^k 0 1^l 0 1^m$

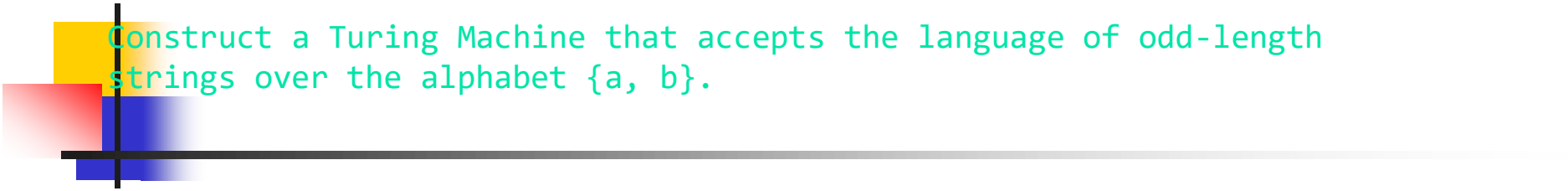

w = 11011
T = 110110110110011010...
w = 11011011011
0 → x1
1 → x2
(T, w) = 000
x2 x2 x1 x2 x2
1 0 1 0 1 0 1 0 1
= 1
w → 110110110111


UL =
⟨T1, w1⟩
⟨T2, w2⟩
⟨T3, w3⟩
⟨Tn, wn⟩
⟨T1, w1⟩
UTM
accept w1 if
T1 accept
reject w1 if
T1 reject


U T M
type
(T, w)
type symbol
states

Old Question 2078: TM encoding (Class-Work)

- Construct a Turing Machine that accepts the language of odd-length strings over the alphabet {a, b}. Give the complete encoding for this TM as well as its input string w = abb in the binary alphabet that is recognized by Universal Turing Machine. [10 marks]
- What is the Universal Turing machine? Explain the working mechanism of the Universal Turing machine for processing the binary code input for (T, w) where T is a specific Turing machine and w is input to T. [10]



Construct a Turing Machine that accepts the language of odd-length strings over the alphabet {a, b}.

Old Question: What do you mean by computational Complexity? Explain the time and space complexity of a Turing machine.

TIME AND SPACE COMPLEXITY OF A TURING MACHINE

- the time complexity: the measure of the number of times the tape moves when the machine is initialized for some input symbols
- the space complexity: the number of cells of the tape written.
- Time complexity all reasonable functions –

  T(n) = O(n log n)
- TM's space complexity –

  S(n) = O(n)