1.75

Undecidability of the Post Correspondence Problem

PROOF:

Approach: Take a problem that is already proven to be undecidable and try to convert it to PCP.
If we can successfully convert it to an equivalent PCP then we prove that PCP is also undecidable.

ACCEPTANCE PROBLEM OF A TURING MACHINE *(This is an undecidable probem)*

$\Downarrow$

Convert this to PCP (called Modified PCP - MPCP)

$\Sigma = \{a,b\}$ $\qquad$ $\Gamma = \{a, b, x, B\}$ $\quad B \to ␣$

$$\left[\frac{\#}{\#}\right]$$

i/p: $w = aba$

Step 1

a	b	a	

$\uparrow$
q_0

$$\frac{\#}{\# q_0 w}$$

$$q_0 aba \Rightarrow \left[\frac{\#}{\# q_0 aba}\right]$$

Step 2: $\delta(q_0, a) = q_1,$

Step2: $\delta(q_0, a) = (q_1, x, R)$

$$q_0 a = x q_1 \Rightarrow \left[\frac{q_0 a}{x q_1}\right]$$

Step3: $\delta(q_1, b) = (q_2, x, L)$

$Y \in \Gamma$

$\Gamma = \{a, b, x, B\}$

$$Y q_1 b = q_2 Y x$$

$$q_0 a = x q_1 \Rightarrow \left[\frac{q_0 a}{x q_1}\right]$$

Step 3: $\delta(q_1, b) = (q_2, x, L)$

$Y \in \Gamma$

$\Gamma = \{a, b, x, B\}$

$$Y q_1 b = q_2 Y x$$

$$\left[\frac{a q_1 b}{q_2 a x}\right], \left[\frac{b q_1 b}{q_2 b x}\right], \left[\frac{x q_1 b}{q_2 x x}\right], \left[\frac{B q_1 b}{q_2 B x}\right]$$

1.75

Step4: For all possible Tape symbols

$\Gamma = \{a, b, x, B\}$

$$\left[\frac{a}{a}\right], \left[\frac{b}{b}\right], \left[\frac{x}{x}\right], \left[\frac{B}{B}\right]$$

Step5: For all possible tape symbols after reaching the accepting state.

Accept q_2

$$\left[\frac{aq_2}{q_2}\right] \left[\frac{q_2a}{q_2}\right], \left[\frac{bq_2}{q_2}\right], \left[\frac{q_2b}{q_2}\right] \left[\frac{xq_2}{q_2}\right] \left[\frac{q_2x}{q_2}\right] \left[\frac{Bq_2}{q_2}\right] \left[\frac{q_2B}{q_2}\right]$$

Step6: $\left[\frac{\#}{\#}\right]$ $\quad \frac{\#}{B\#}$

1.75

$$\left[\frac{a}{a}\right], \left[\frac{b}{b}\right], \left[\frac{X}{X}\right], \left[\frac{B}{B}\right]$$

Step 5: For all possible tape symbols after reaching the accepting state.
Accept q_2

$$\left[\frac{a q_2}{q_2}\right] \left[\frac{q_2 a}{q_2}\right], \left[\frac{b q_2}{q_2}\right], \left[\frac{q_2 b}{q_2}\right] \left[\frac{X q_2}{q_2}\right] \left[\frac{q_2 X}{q_2}\right] \left[\frac{B q_2}{q_2}\right] \left[\frac{q_2 B}{q_2}\right]$$

Step 6: $\left[\frac{\#}{\#}\right] \quad \left[\frac{\#}{B\#}\right]$

Step 7: $\left[\frac{q_2 \#\#}{\#}\right]$

Solution:

$$\left[\frac{\cancel{\#}}{\cancel{\#}\,\cancel{q_0}\,\cancel{a}\,\cancel{b}\,\cancel{a}\,\#}\right]\left[\frac{\cancel{q_0a}}{xq_1}\right]\left[\frac{\cancel{b}}{b}\right]\left[\frac{\cancel{a}}{a}\right]\left[\frac{\#}{\#}\right]$$

$$\left[\frac{\cancel{\#}}{\#\ \cancel{x}\,\cancel{q_1}\,\cancel{b}\,\cancel{a}\ \#}\right]\left[\frac{\cancel{xq_1b}}{q_2xx}\right]\left[\frac{\cancel{a}}{a}\right]\left[\frac{\#}{\#}\right]$$

$$\left[\frac{\cancel{\#}}{\#\ \cancel{q_2}\ \cancel{x}\,\cancel{x}\,\cancel{a}\,\#}\right]\left[\frac{\cancel{q_2x}}{q_2}\right]\left[\frac{\cancel{x}}{x}\right]\left[\frac{\cancel{a}}{a}\right]\left[\frac{\#}{\#}\right]$$

$$\left[\frac{\#}{\cancel{\#}\ \cancel{q_2}\,\cancel{x}\,a\ \#}\right]\left[\frac{\cancel{q_2x}}{q_2}\right]\left[\frac{\cancel{a}}{a}\right]\left[\frac{\#}{\#}\right]$$

$$\left[\frac{\#}{\#\ q_2\,a\,\#}\right]$$

$$\left[\frac{\cancel{\#}}{\cancel{\#}\ \cancel{x}\ \cancel{q_1}\cancel{b}\cancel{a}\ \#}\right]\left[\frac{\cancel{x q_1 b}}{q_2 x x}\right]\left[\frac{\cancel{a}}{a}\right]\left[\frac{\#}{\#}\right]$$

$$\left[\frac{\cancel{\#}}{\cancel{\#}\ \cancel{q_2}\ \cancel{x}\cancel{x}\cancel{a}\#}\right]\left[\frac{\cancel{q_2 x}}{q_2}\right]\left[\frac{\cancel{x}}{x}\right]\left[\frac{\cancel{a}}{a}\right]\left[\frac{\#}{\#}\right]$$

$$\left[\frac{\#}{\cancel{\#}\ \cancel{q_2}\cancel{x}a\ \#}\right]\left[\frac{\cancel{q_2 x}}{q_2}\right]\left[\frac{\cancel{a}}{a}\right]\left[\frac{\#}{\#}\right]$$

$$\left[\frac{\#}{\#\ \cancel{q_2}\cancel{a}\#}\right]\left[\frac{\cancel{q_2 a}}{q_2}\right]\left[\frac{\#}{\#}\right]$$

$$\left[\frac{\cancel{\#}}{\cancel{\#}\ \cancel{q_2}\cancel{\#}}\right]\left[\frac{\cancel{q_2 \# \#}}{\cancel{\#}}\right]\ \checkmark$$

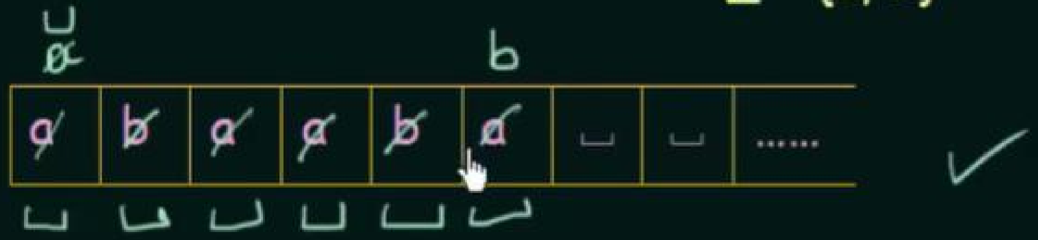
PCP is <u>undecidable</u>

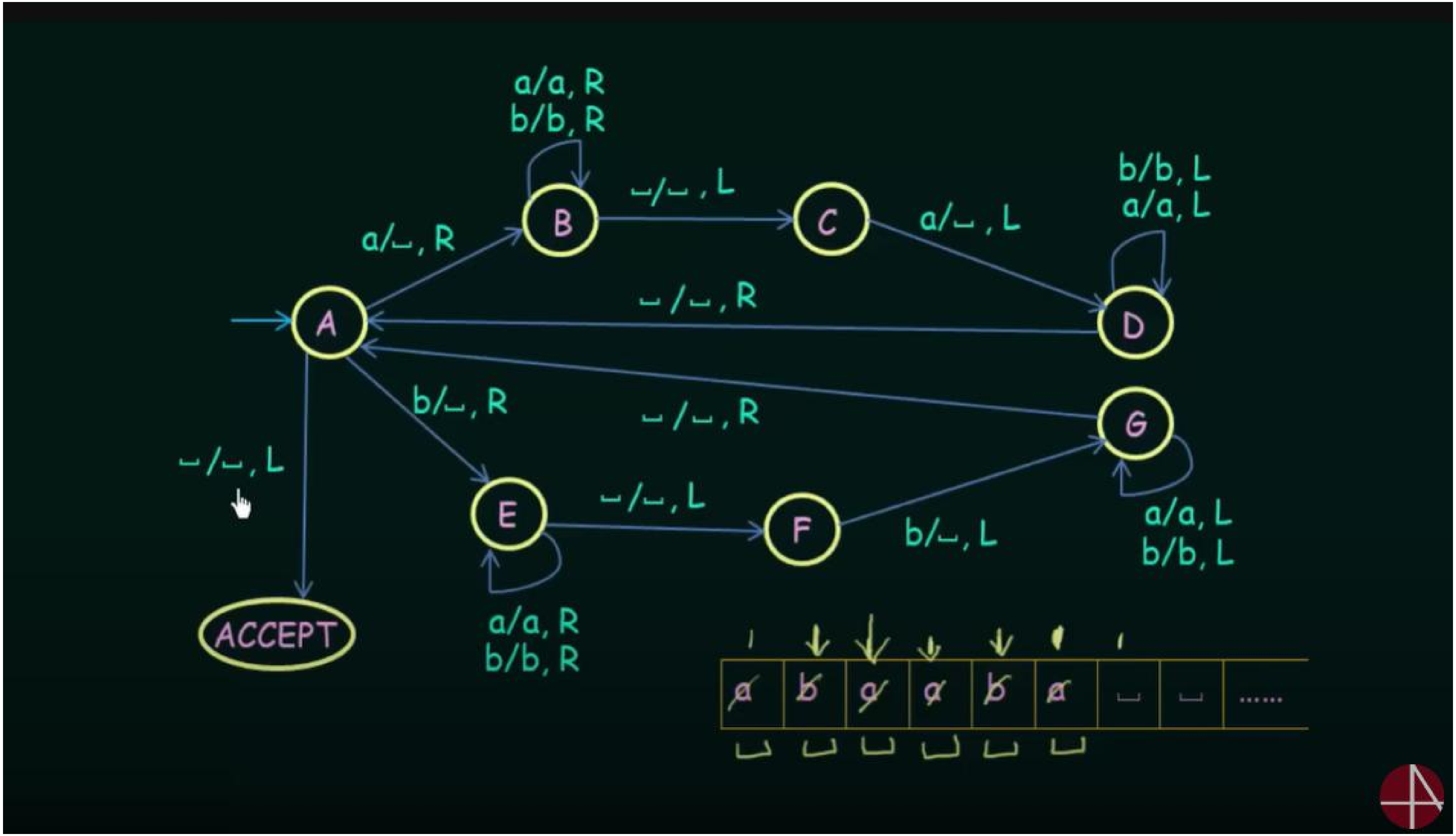
Turing Machine for Even Palindromes

Design a Turing Machine that accepts Even Palindromes over the alphabet

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

Eg.



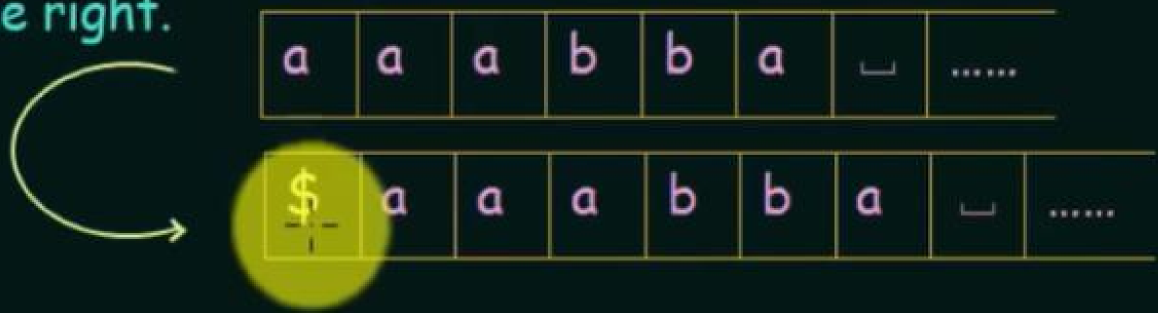


a/a, R
b/b, R
a/␣, R
B
␣/␣, L
C
a/␣, L
b/b, L
a/a, L
A
␣/␣, R
D
b/␣, R
␣/␣, R
G
␣/␣, L
E
␣/␣, L
F
b/␣, L
a/a, L
b/b, L
ACCEPT
a/a, R
b/b, R
a b a a b a ␣ ␣ ......

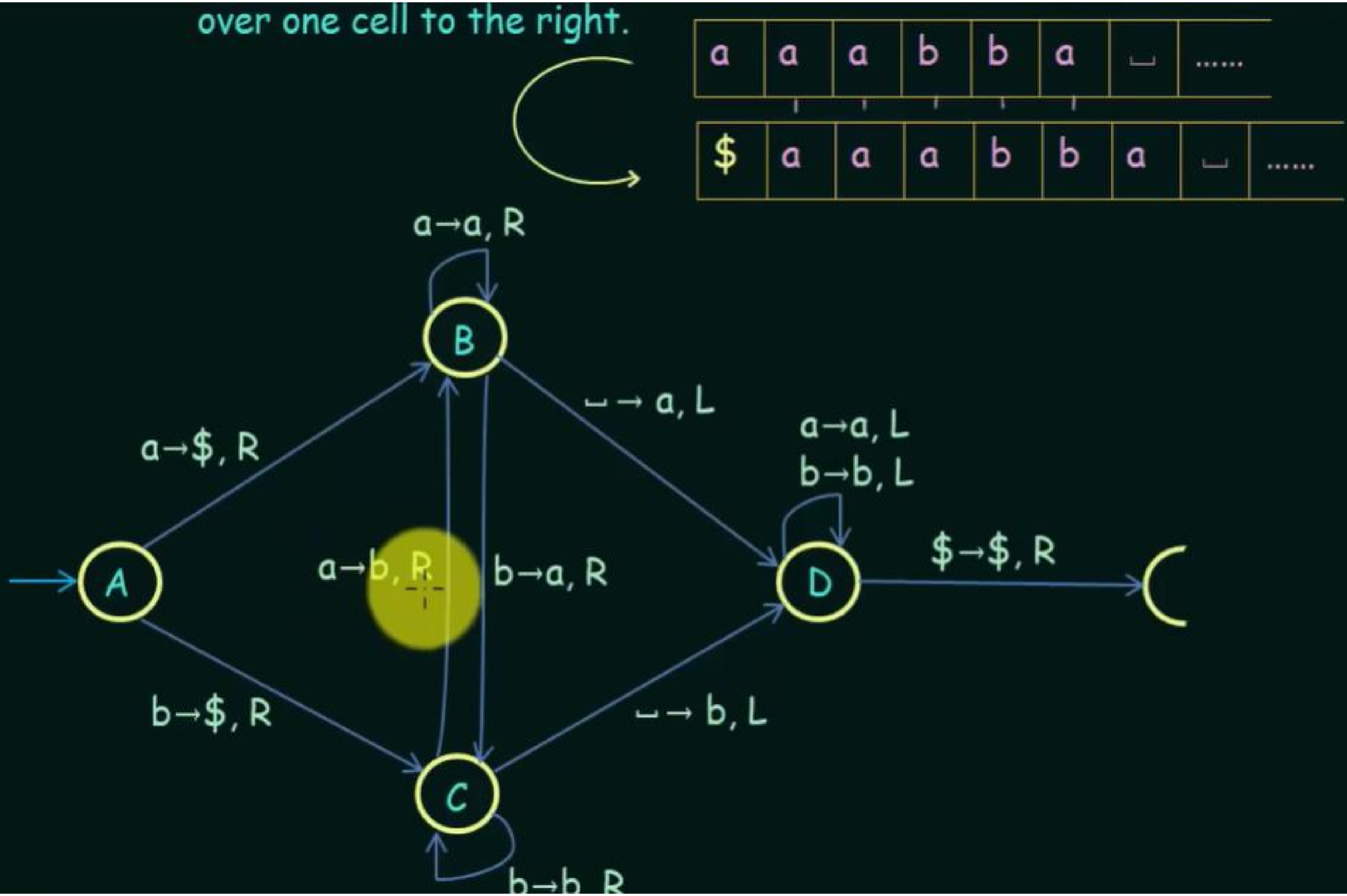
Turing Machine Programming Techniques (Part-1)

Problem: How can we recognize the left end of the Tape of a Turing Machine ?

Solution: Put a Special Symbol $ on the left end of the Tape and shift the input over one cell to the right.



over one cell to the right.



Turing Machine Programming Techniques (Part-2)

Example: Build a Turing Machine to recognize the language $0^N1^N0^N$

IDEA

We already have a Turing Machine to turn 0^N1^N to x^Ny^N and to decide that language.

USE THIS TURING MACHINE AS A SUBROUTINE

Step 1: 00000 11111 00000

⇩

xxxxx yyyyy 00000

Step 2: Build a similar Turing Machine to recognize y^N0^N

Step 3: Build the final Turing Machine by combining these two smaller Turing Machines together into one larger Turing Machine

Turing Machine Programming Techniques (Part-3)

COMPARING TWO STRINGS

A Turing Machine to decide $\{ w \# w \mid w \in \{a,b,c\}^* \}$

Solution:

- Use a new symbol such as 'x'
- Replace each symbol into an x after it has been examined

Problem:

Can we do it non-destructively? i.e. without loosing the original strings?

Solution:

Replace each unique symbol with another unique symbol instead of replacing all with the same symbol

Eg. $a \rightarrow p$

$b \rightarrow q$

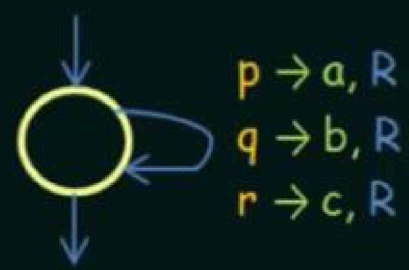
$c \rightarrow r$

a b b a c # a b b a c

$\Downarrow$

p q q p r # p q q p r

Restore the original strings if required



2.00

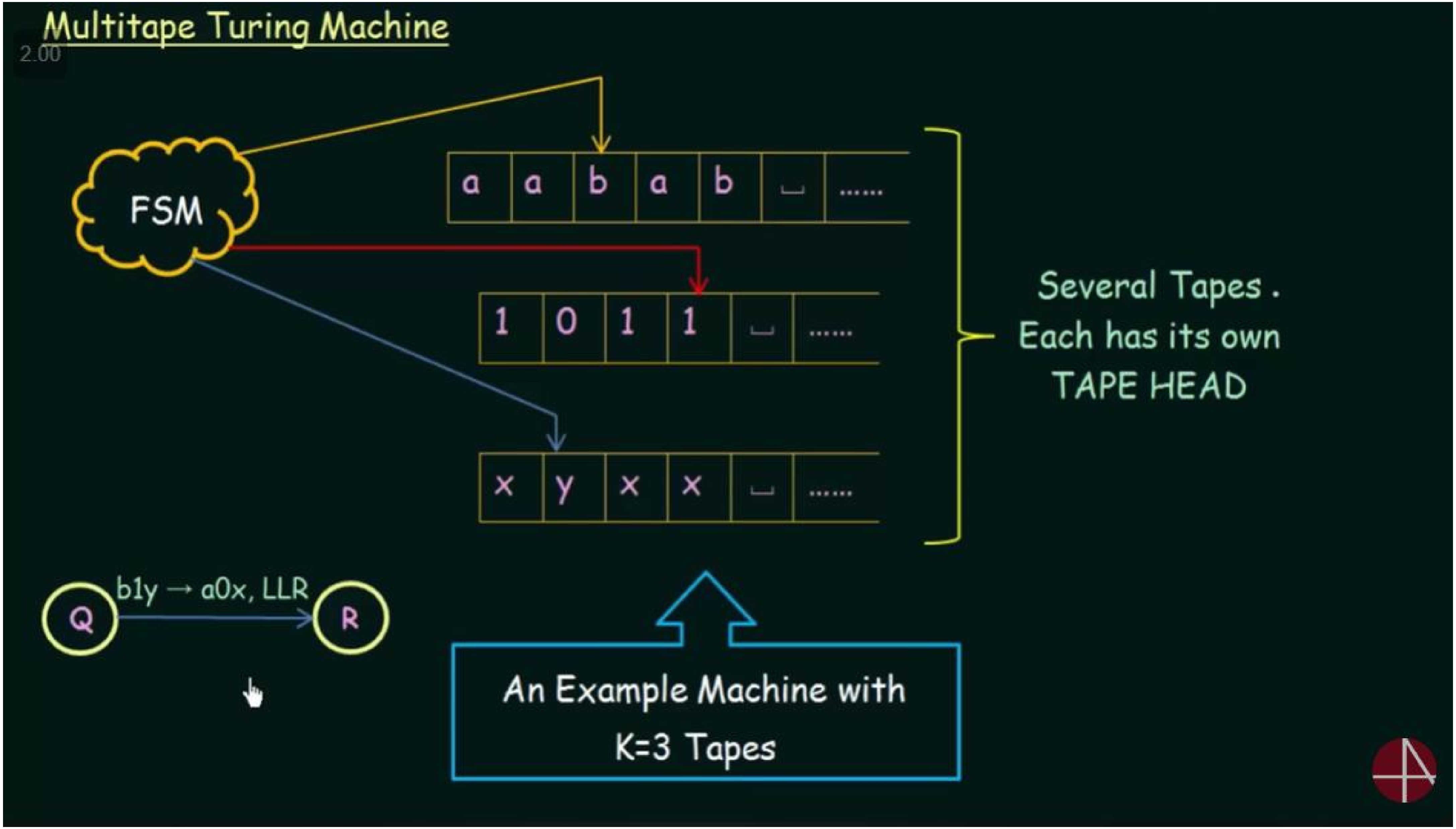
Multitape Turing Machine

Theorem: Every Multitape Turing Machine has an equivalent Single Tape Turing Machine

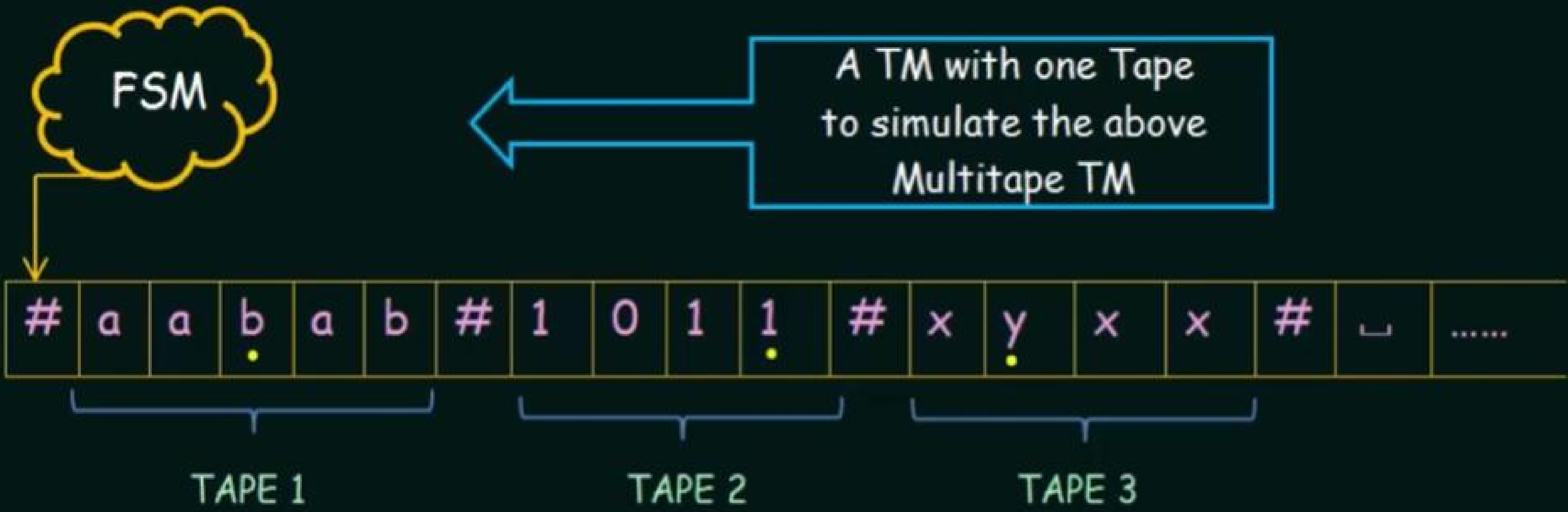
Proof

Given a Multitape Turing Machine show how to build a single tape Turing Machine

- Need to store all tapes on a single tape
  Show data representation
- Each tape has a tape head
  Show how to store that info
- Need to transform a move in the Multitape TM into one or moves in the Single Tape TM


Multitape Turing Machine
FSM
a a b a b ␣ ......
1 0 1 1 ␣ ......
x y x x ␣ ......
Several Tapes .
Each has its own
TAPE HEAD
Q
b1y → a0x, LLR
R
An Example Machine with
K=3 Tapes

Single Tape Turing Machine



- Add "dots" to show where Head "K" is
- To simulate a transition from state Q, we must scan our Tape to see which symbols are under the K Tape Heads
- Once we determine this and are ready to MAKE the transition, we must scan across the tape again to update the cells and move the dots
- Whenever one head moves off the right end, we must shift our tape so we can insert a ␣

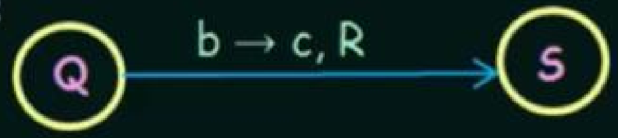
Nondeterminism in Turing Machine (Part-1)

Nondeterministic Turing Machines:

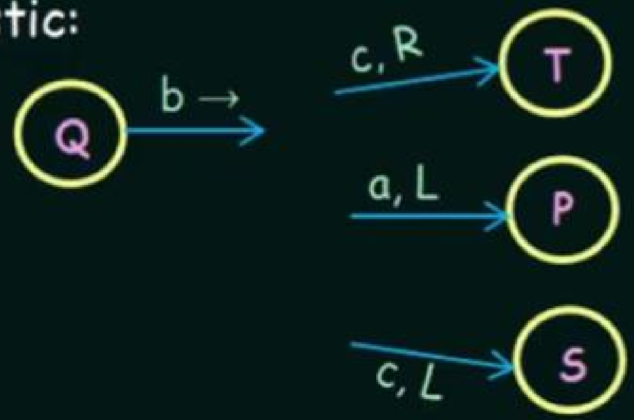
Transition Function:

$$\delta : \quad Q \times \Sigma \rightarrow P\{\Gamma \times (R/L) \times Q\}$$

Deterministic:

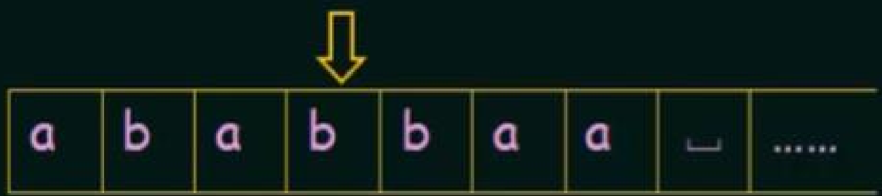


Nondeterministic:

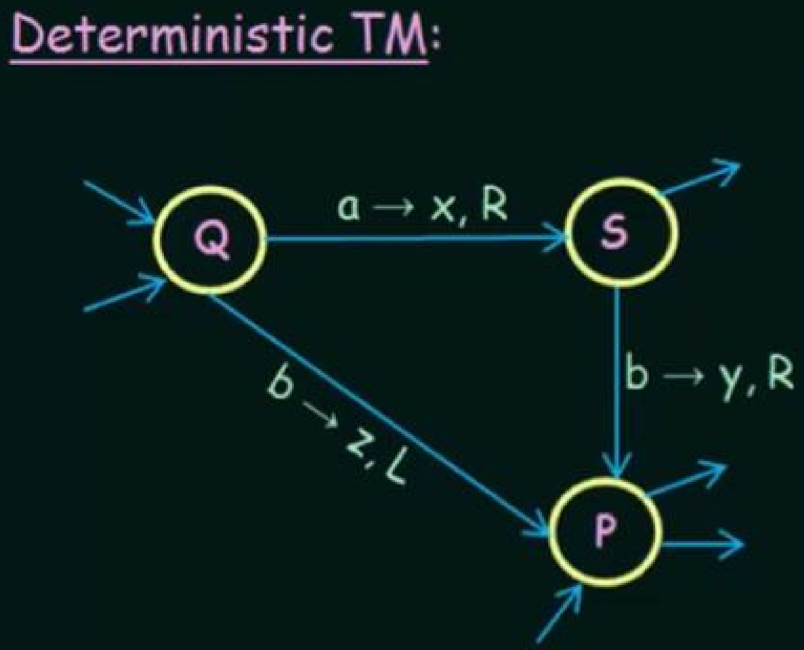


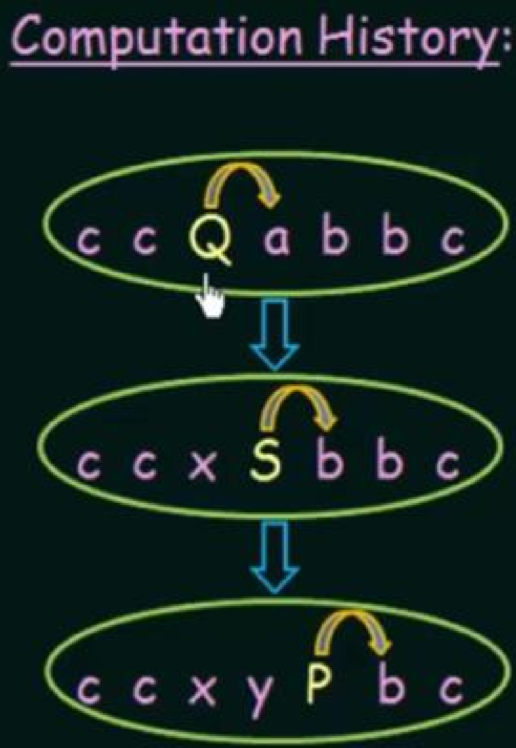
CONFIGURATION

- A way to represent the entire state of a TM at a moment during computation
- A string which captures:
  - The current state
  - The current position of the Head
  - The entire Tape contents



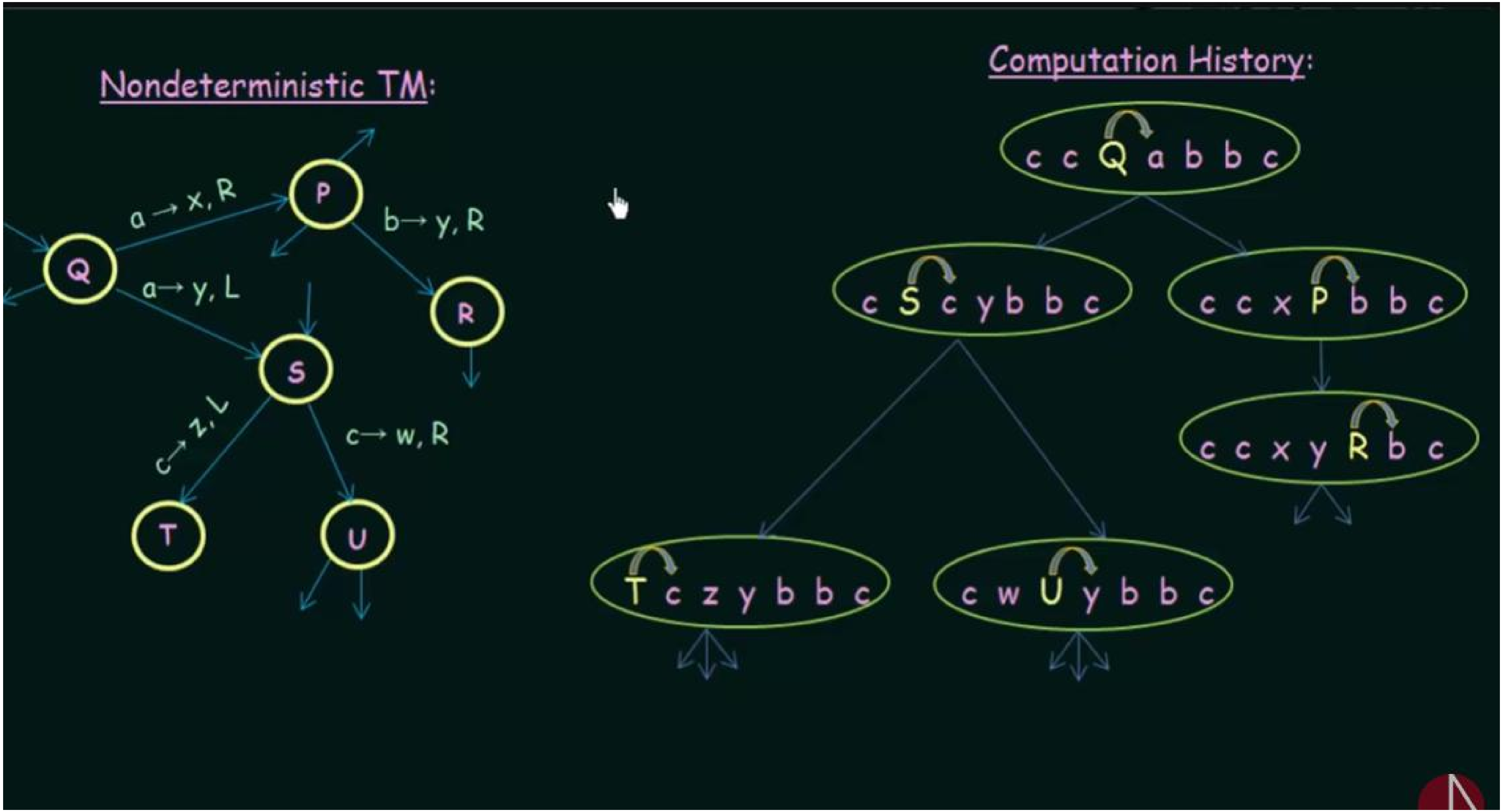
a b a Q b b a a





With Nondeterminism:

At each moment in the computation there can be more than one successor configuration


Nondeterministic TM:
a → x, R
b → y, R
a → y, L
c → z, L
c → w, R
Q
P
R
S
T
U
Computation History:
c c Q a b b c
c S c y b b c
c c x P b b c
c c x y R b c
T c z y b b c
c w U y b b c

Outcomes of a Nondeterministic Computation:

ACCEPT If any branch of the computation accepts, then the nondeterministic TM will Accept.

REJECT If all branches of the computation HALT and REJECT (i.e. no branches accept, but all computations HALT) then the Nondeterministic TM Rejects.

LOOP Computation continues but ACCEPT is never encountered. Some branches in the computation history are infinite.

Nondeterminism in Turing Machine (Part-2)

2.00

Theorem: Every Nondeterministic TM has an equivalent Deterministic TM

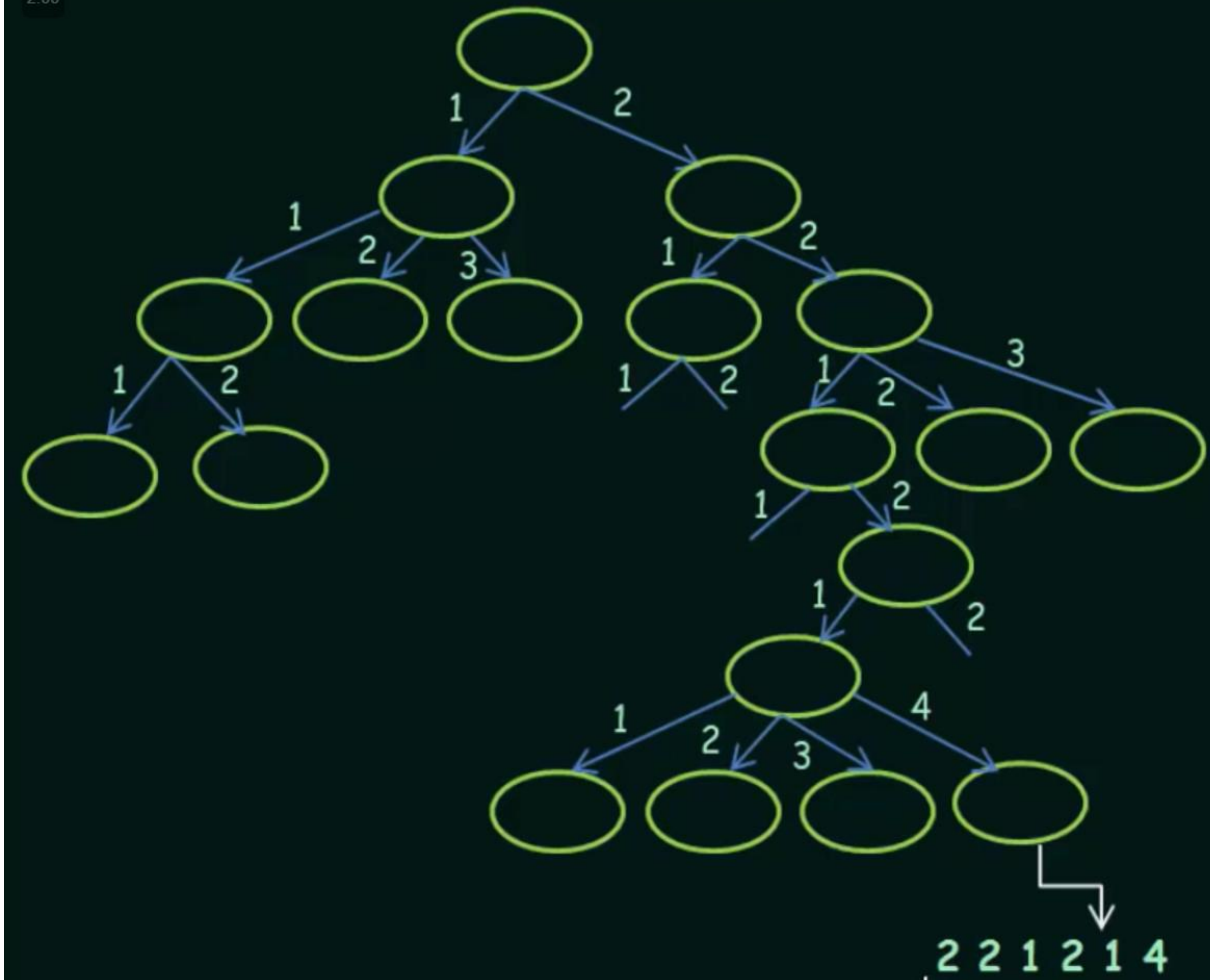
Proof:

- Given a Nondeterministic TM (N) show how to construct an equivalent Deterministic TM (D)
- If N accepts on any branch, the D will Accept
- If N halts on every branch without any ACCEPT, then D will Halt and Reject.

Approach:

- Simulate N
- Simulate all branches of computation
- Search for any way N can Accept

Computational History:



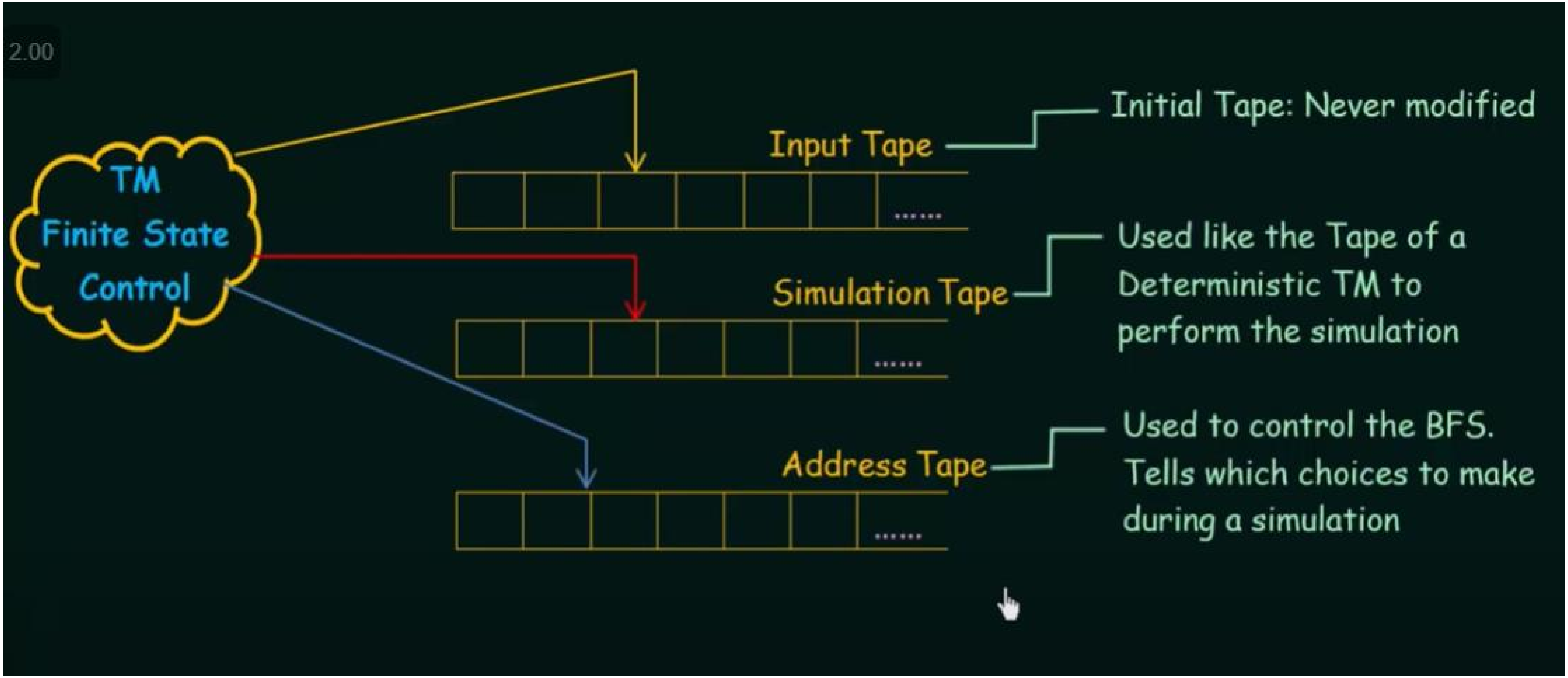
- A Path to any Node is given by a number
- Search the tree looking for ACCEPT

Search Order:

- DEPTH FIRST SEARCH ✗
- BREADTH FIRST SEARCH ✓

To examine a node:

- Perform the entire computation from scratch
- The path numbers tells which of the many nondeterministic choices to make


2.00
TM
Finite State
Control
Input Tape
Initial Tape: Never modified
Simulation Tape
Used like the Tape of a
Deterministic TM to
perform the simulation
Address Tape
Used to control the BFS.
Tells which choices to make
during a simulation

Algorithm:

2.00

Initially: TAPE 1 contains the Input
TAPE 2 and TAPE 3 are empty

- Copy TAPE 1 to TAPE 2
- Run the Simulation
- Use TAPE 2 as "The Tape"
- When choices occur (i.e. when Nondeterministic branch points are encountered) consult TAPE 3
- TAPE 3 contains a Path. Each number tells which choice to make
- Run the Simulation all the way down the branch as far as the address/path goes (or the computation dies)
- Try the next branch
- Increment the address on TAPE 3
- REPEAT

If ACCEPT is ever encountered, Halt and Accept
If all branches Reject or die out, then Halt and Reject

Turing Machine as Problem Solvers

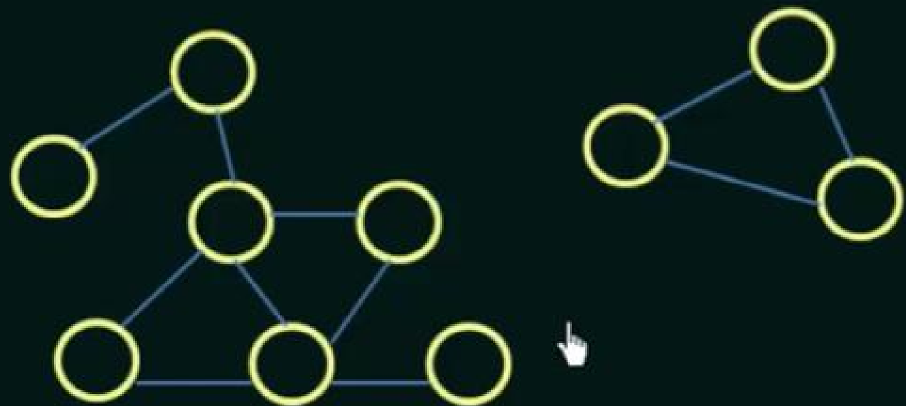
Any arbitrary Problem can be expressed as a language

-Any instance of the problem is encoded into a string

The string is in the language $\Rightarrow$ The answer is YES

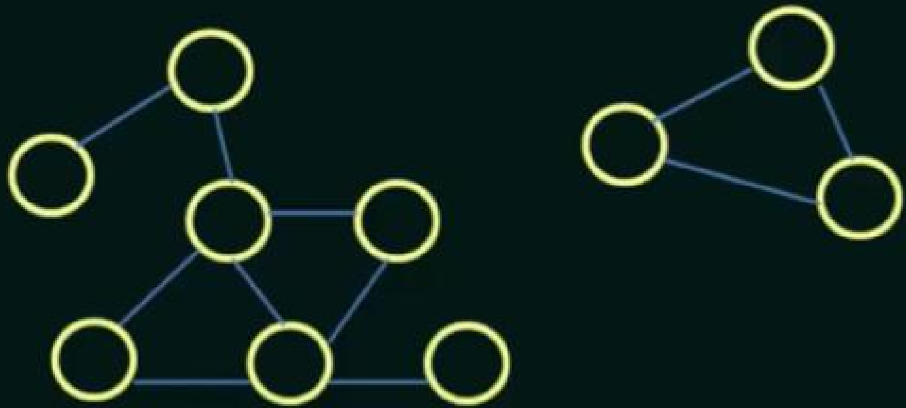
The string is not in the language $\Rightarrow$ The answer is NO

Example: Is this undirected graph connected?



2.00

Example: Is this undirected graph connected?



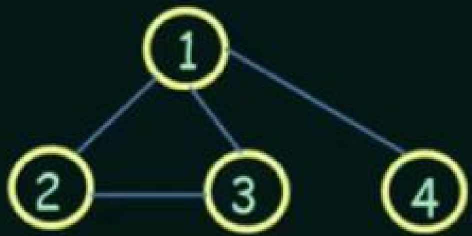
We must encode the problem into a language.

$A = \{ \langle G \rangle \mid G \text{ is a connected graph} \}$

We would like to find a TM to decide this language:

ACCEPT	=	"YES", This is a connected graph
REJECT	=	"NO", This is not a connected graph / or this is not a valid Representation of a graph.
LOOP	=	This problem is decidable. Our TM will always halt

Representation of Graph:



$$\langle G\rangle = \underbrace{(1,2,3,4)}_{\text{List of nodes}}\ \underbrace{(\,(1,2),(2,3),(1,3),(1,4)\,)}_{\text{Edges}}$$

$$\Sigma = \{\ (,\ ),\ ,,\ 1,\ 2,\ 3,\ 4,\ \ldots\ldots,0\ \}$$

(	1	,	2	,	3	,	4	,	....	)	)	␣	.....

High Level Algorithm:

```
Select a Node and Mark it
REPEAT
    For each node N
        If N is unmarked and there is an edge from N to an already marked
        node
    Then
        Mark Node N
    End

Until no more nodes can be marked
    For each Node N
        If N is unmarked
        Then REJECT
    End
ACCEPT
```

Implementation Level Algorithm:

- Check that input describes a valid graph
- Check Node List
    - Scan "(" followed by digits ...
    - Check that all nodes are different i.e. no repeats
    - Check edge lists ...

      etc.

- Mark First Node
    - Place a dot under the first node in the node list
    - Scan the node list to find a node that is not marked

      etc.

Decidability and Undecidability

Recursive Language:

- 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.

Recursively Enumerable Language:

- 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'.

Decidable Language:

A language 'L' is decidable if it is a recursive language. All decidable languages are recursive languages and vice-versa.

Partially Decidable Language:

A language 'L' is partially decidable if 'L' is a recursively enumerable language.

Undecidable Language:

- A language is undecidable if it is not decidable.
- An undecidable language may sometimes be partially decidable but not decidable.
- If a language is not even partially decidable, then there exists no Turing machine for that language

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

The Universal Turing Machine

The Language

$A_{TM} = \{ \langle M,w \rangle \mid M \text{ is a Turing Machine and } M \text{ accepts } w\}$

is Turing Recognizable

Given the description of a TM and some input, can we determine whether the machine accepts it?

- Just Simulate/ Run the TM on the input

M Accepts w: Our Algorithm will Halt & Accept

M Rejects w: Our Algorithm will Halt & Reject.

M Loops on w: Our Algorithm will not Halt.

The Universal Turing Machine

Input: M = the description of some TM

w = an input string for M

Action: - Simulate M

- Behave just like M would (may accept, reject or loop)

The UTM is a recognizer (but not a decider) for

A_{TM} = { <M,w> | M is a TM and M accepts w}

The Halting Problem

Given a Program, WILL IT HALT ?

Given a Turing Machine, will it halt when run on some particular given input string?

Given some program written in some language (Java/C/ etc.) will it ever get into an infinite loop or will it always terminate?

The Halting Problem

Given a Program, WILL IT HALT ?

Given a Turing Machine, will it halt when run on some particular given input string?

Given some program written in some language (Java/C/ etc.) will it ever get into an infinite loop or will it always terminate?

Answer:

- In General we can't always know.
- The best we can do is run the program and see whether it halts.
- For many programs we can see that it will always halt or sometimes loop

BUT FOR PROGRAMS IN GENERAL THE QUESTION IS UNDECIDABLE.

Undecidability of the Halting Problem

Given a Program, WILL IT HALT ?

Can we design a machine which if given a program can find out or decide if that program will always halt or not halt on a particular input?

Let us assume that we can:

H (P, I)

- Halt
- Not Halt

This allows us to write another Program:

```
C ( X )
  if { H (X, X) == Halt }
      Loop Forever;
  else
      Return;
```

Let us assume that we can:

H (P, I)
- Halt
- Not Halt

This allows us to write another Program:

```
C ( X )
  if { H (X, X) == Halt }
      Loop Forever;
  else
      Return;
```

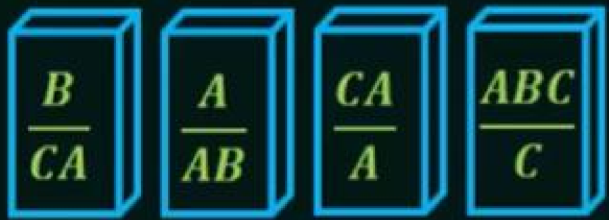
<u>If we run 'C' on itself:</u>

C (C)

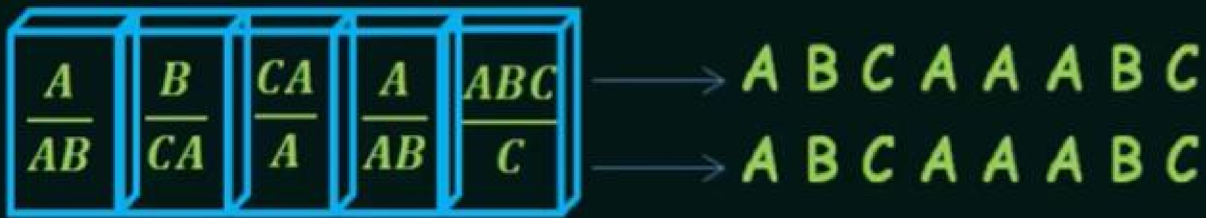
H (C,C) == Halt	H (C,C) == Not Halt
↓	↓
Not Halt	Halt

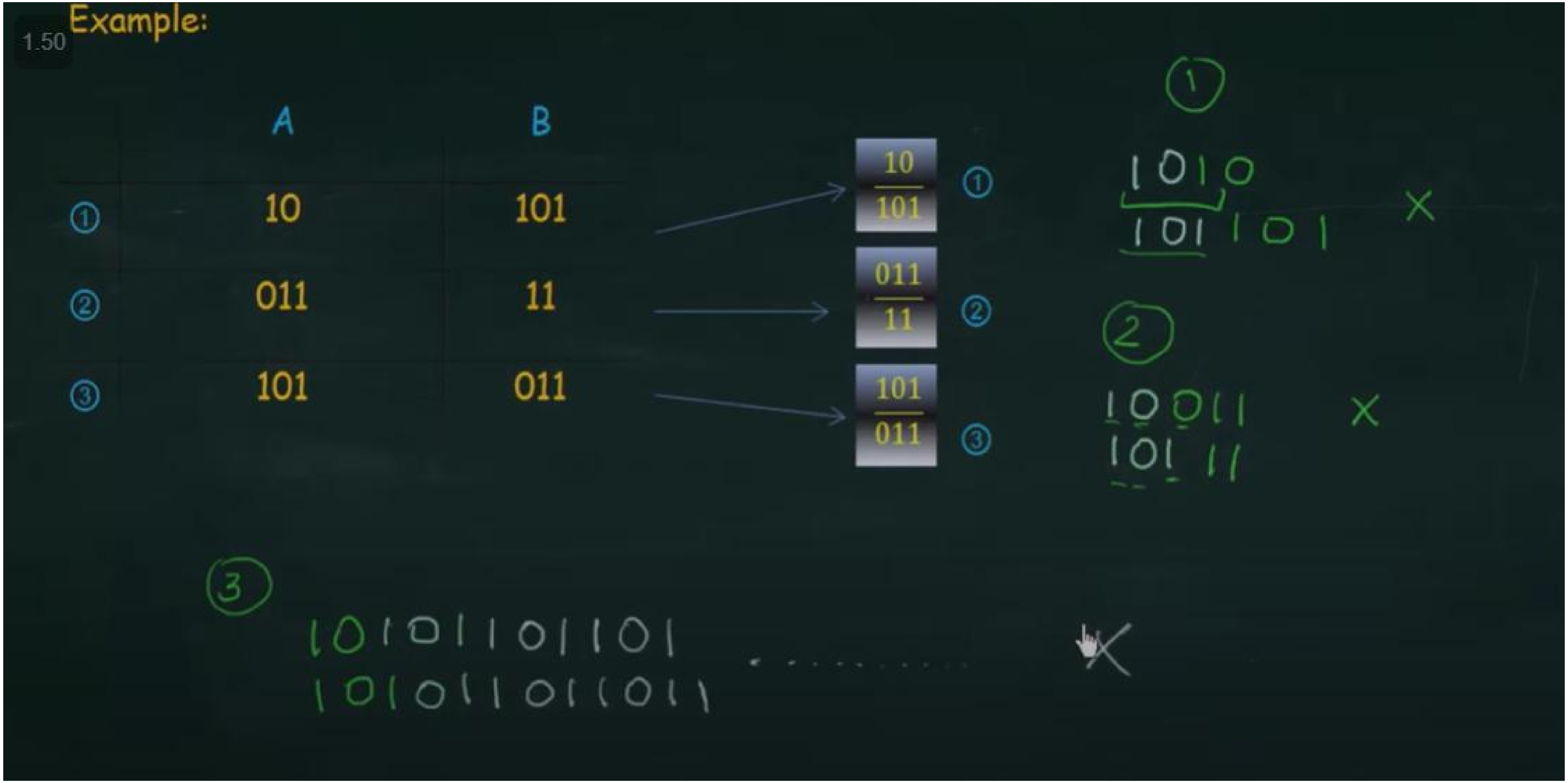
The Post Correspondence Problem

Dominos:



We need to find a sequence of dominos such that the top and bottom strings are the same.




1.50
Example:
A B
① 10 101
② 011 11
③ 101 011
10
101
①
011
11
②
101
011
③
①
1010
101101
X
②
10011
10111
X
③
1010110110101
10101101011011
X