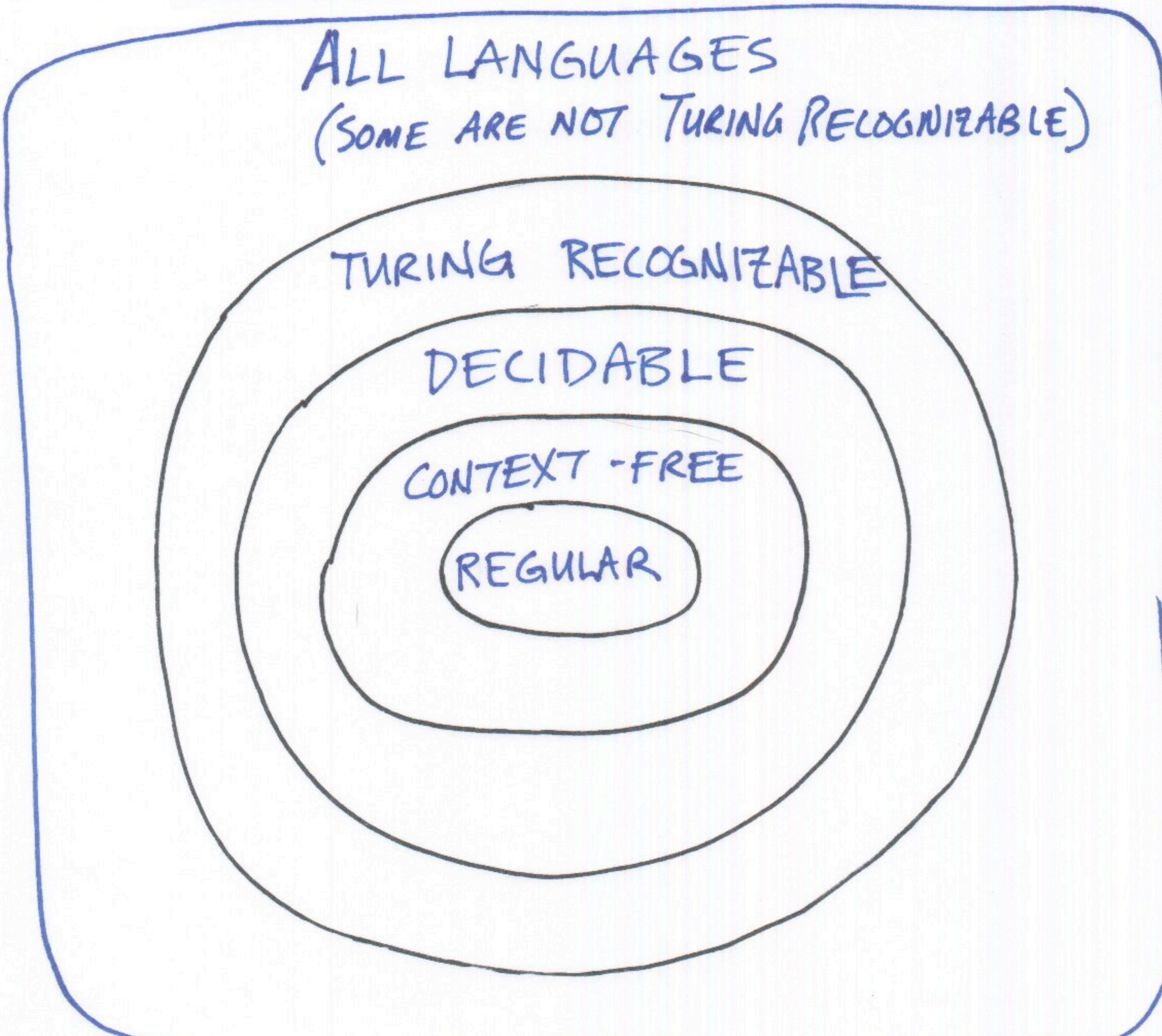
TURING MACHINES

* FINITE STATE MACHINES
  REGULAR LANGUAGES
* PUSHDOWN AUTOMATA
  CONTEXT-FREE LANGUAGES
* TM: TURING MACHINES

A new model of computation.
Not much more elaborate.
A "model" for <u>all</u> computers.

"DECIDABLE" LANGUAGES
"TURING RECOGNIZABLE" LANGUAGES
LANGUAGES THAT ARE
"NOT TURING RECOGNIZABLE."

CLASSES OF LANGUAGES
(THE "LANGUAGE ONION")



THIS IS A *VENN DIAGRAM*
THE SUBSET RELATIONSHIPS
ARE ALL "PROPER SUBSET"

TURING MACHINE DEFINITION

NOTE: THERE ARE VARIATIONS IN THE EXACT DEFINITION FROM TEXTBOOK TO TEXTBOOK.

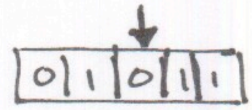
ALL VARIATIONS ARE EQUIVALENT!

WE'LL DISCUSS THIS LATER.

DATA STRUCTURE:

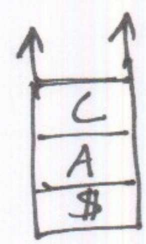
FSM

- THE INPUT STRING



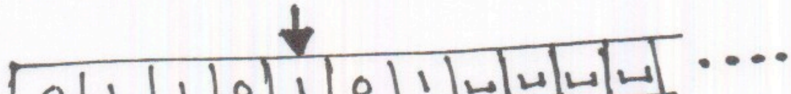
PDA

- THE INPUT STRING
- A STACK



TM

- A "TAPE"



SYMBOLS FROM AN ALPHABET Σ

A SPECIAL BLANK SYMBOL ␣

INFINITE IN ONE DIRECTION
- BUT FILLED WITH BLANKS.

CURRENT POSITION

TAPE ALPHABET

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

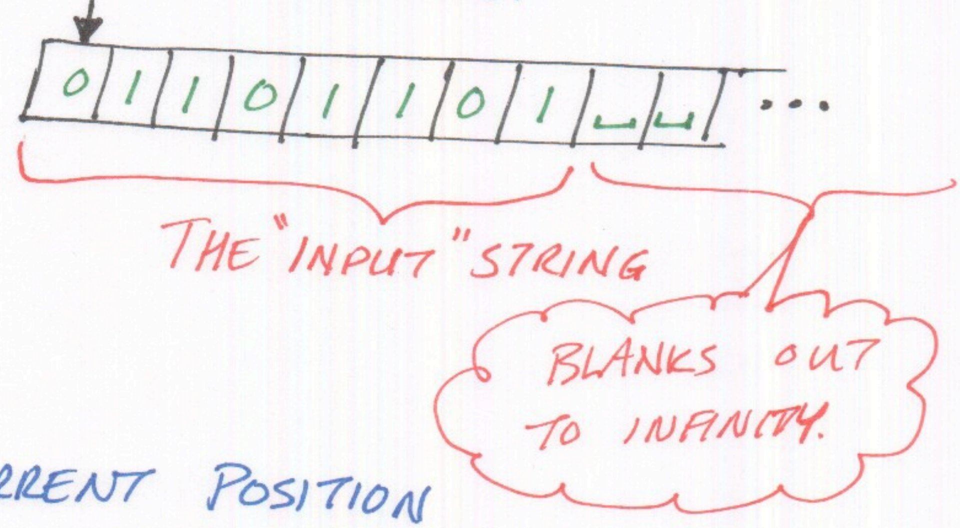
BUT ALSO COMMON:

$$\Sigma = \{0, 1, a, b, x, \#, \$\}$$

THE "BLANK" SYMBOL IS SPECIAL

$$\sqcup \notin \Sigma$$

INITIAL CONFIGURATION:



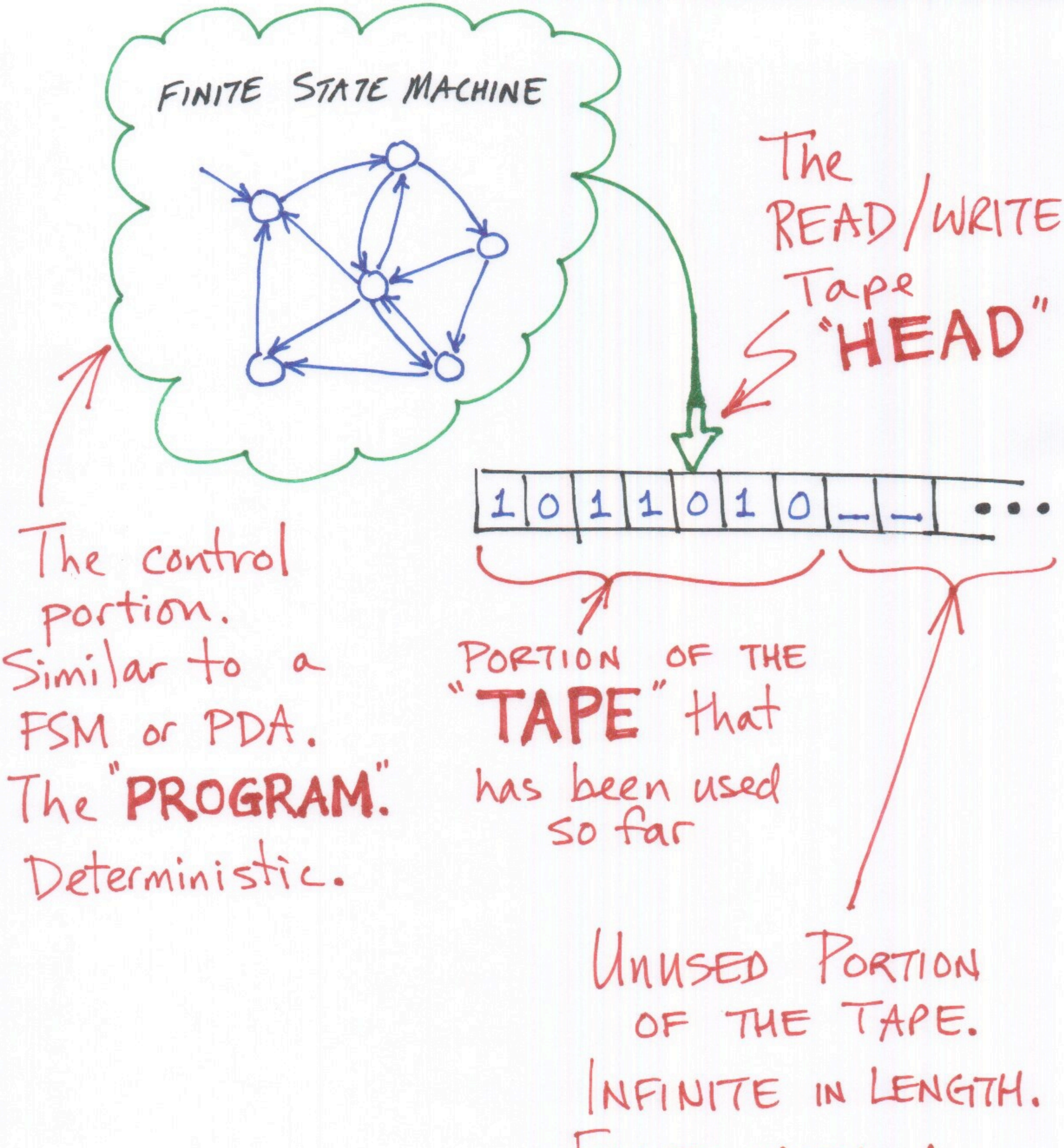
THE CURRENT POSITION
("THE TAPE HEAD")

INITIALLY AT THE LEFTMOST CELL.

CAN MOVE LEFT OR RIGHT.

CAN READ ("SCAN") THE CURRENT SYMBOL

CAN WRITE THE CURRENT SYMBOL

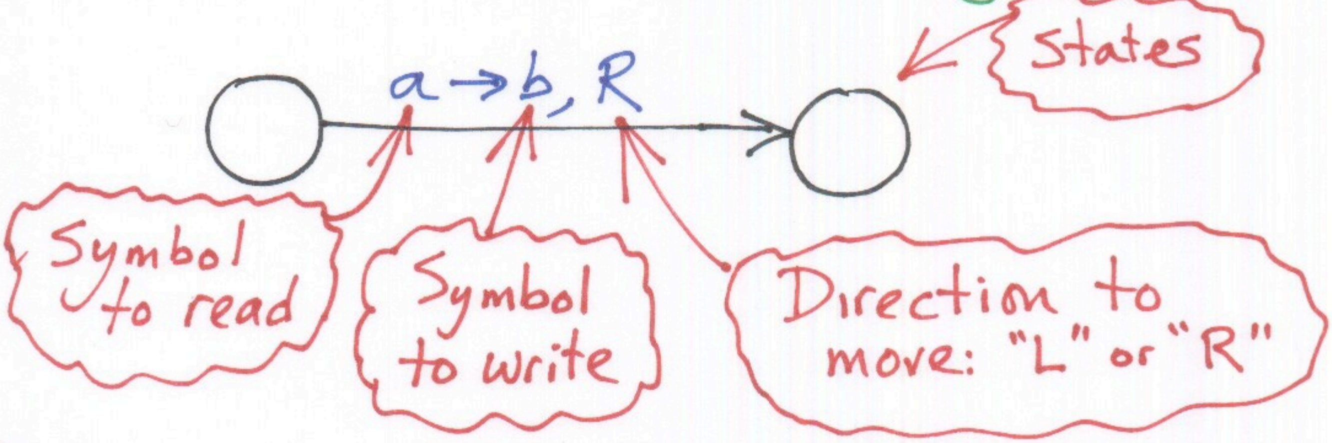

FINITE STATE MACHINE
The READ/WRITE Tape "HEAD"
1 0 1 1 0 1 0 _ _ ...
The control portion. Similar to a FSM or PDA. The "PROGRAM". Deterministic.
PORTION OF THE "TAPE" that has been used so far
UNUSED PORTION OF THE TAPE. INFINITE IN LENGTH. FILLED WITH A SPECIAL "BLANK" SYMBOL _

RULES OF OPERATION

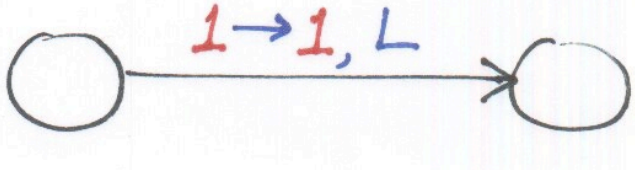
AT EACH STEP OF THE COMPUTATION:

- READ THE CURRENT SYMBOL
- UPDATE (i.e., write) THE SAME CELL.
- MOVE EXACTLY ONE CELL EITHER <u>LEFT</u> OR <u>RIGHT</u>.

(If we are at the left end of the tape and trying to move left, then do not move; stay at left end.)



Don't want to update the cell?
Just write the same symbol.



RULES OF OPERATION - 2

- CONTROL IS WITH A SORT OF FINITE STATE MACHINE.
- INITIAL STATE
- FINAL STATES

  THE "ACCEPT" STATE
  THE "REJECT" STATE

  Exaclty two final states.

- COMPUTATION CAN...

  HALT AND "**ACCEPT**"
  (Whenever the machine enters the ACCEPT state, computation immediately HALTS.)

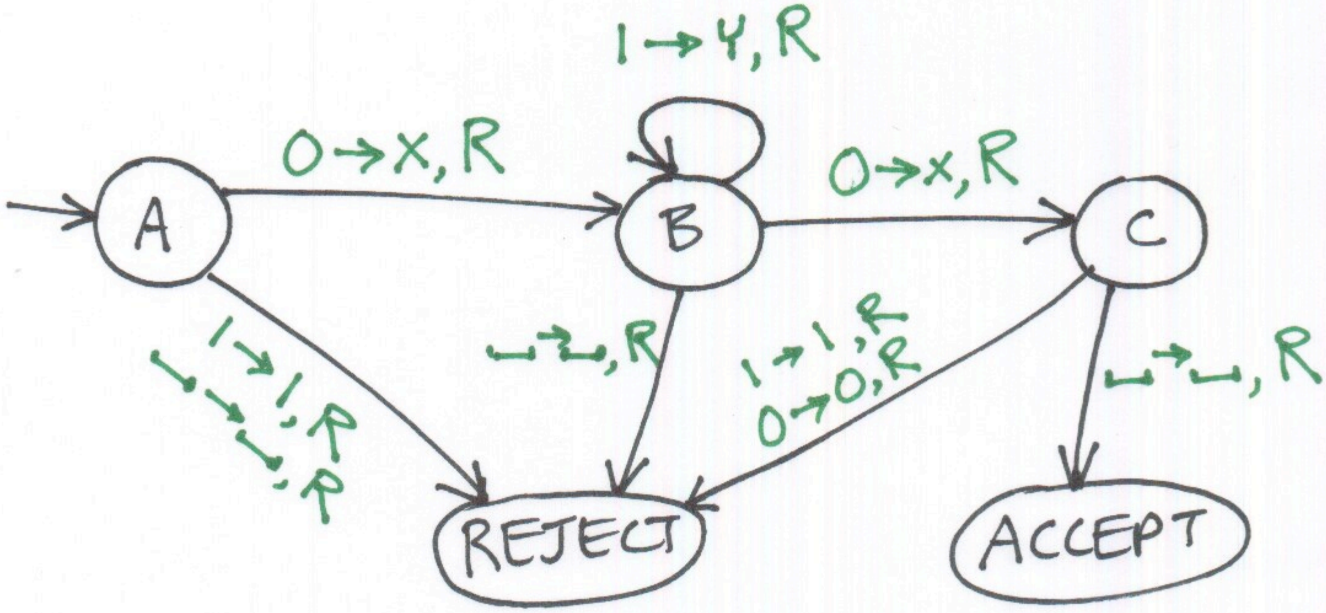
  HALT AND "**REJECT**"
  (Whenever the machine enters the REJECT state, computation immediately HALTS.)

  "**LOOP**"
  (The machine fails to HALT.)

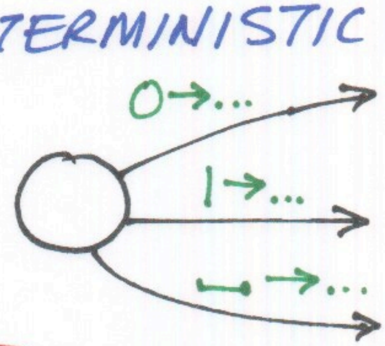
- THE TM IS **DETERMINISTIC**.

AN EXAMPLE

$L = 01^*0$



IS IT DETERMINISTIC?



HENCEFORTH:

IF AN EDGE IS MISSING...
ASSUME IT LEADS TO REJECT.

EXAMPLE

$L = 0^N 1^N$

INPUT ALPHABET:

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

ALGORITHM

CHANGE "0" TO "X"

MOVE RIGHT TO FIRST "1"

IF NONE: REJECT.

CHANGE "1" INTO "Y"

MOVE LEFT TO LEFTMOST "0"

REPEAT UNTIL NO MORE "0"S

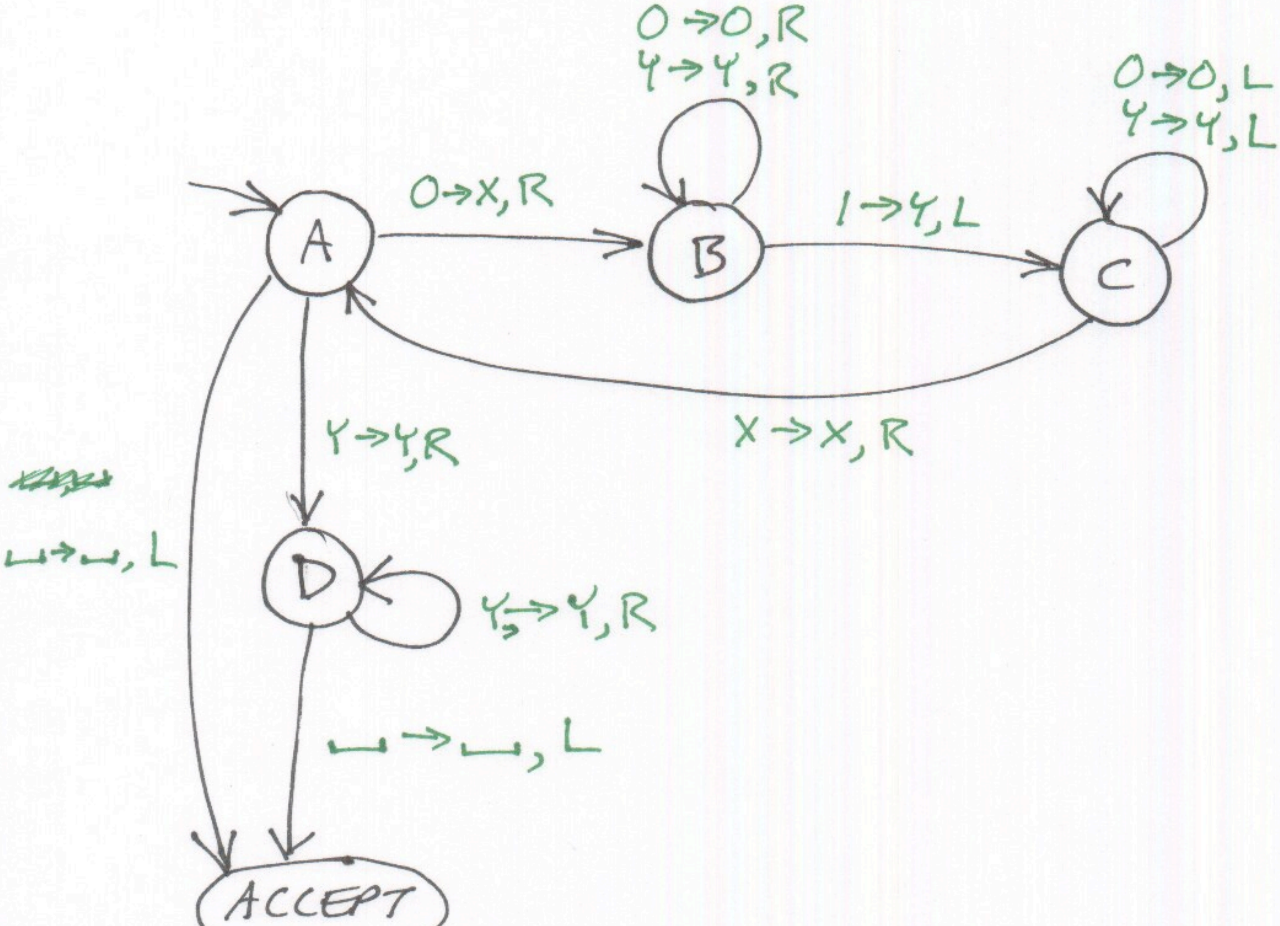
MAKE SURE NO MORE "1"S REMAIN

A COMPUTATION HISTORY

0	0	0	0	1	1	1	1
X	0	0	0	1	1	1	1
X	0	0	0	Y	1	1	1
X	X	0	0	Y	1	1	1
			⋮				
X	X	X	X	Y	Y	Y	Y

TAPE ALPHABET:

$\Gamma = \{0, 1, X, Y, \sqcup\}$



QUESTION:

Is this machine correct?
Does it work?
Does it contain bugs?

TM's model computers.
In this way they are similar!

$\boxed{\text{DETAILS}}$

$$(Q, \Sigma, \Gamma, \delta, q_0, q_{ACCEPT}, q_{REJECT})$$

Q = Set of states.

OFTEN WE NEED A FEW EXTRA SYMBOLS TO MAKE OUR COMPUTATION EASIER.

Σ = INPUT ALPHABET.

Γ = TAPE ALPHABET $\quad \Sigma \subseteq \Gamma$

THE INPUT CANNOT CONTAIN A BLANK.

$\sqcup \notin \Sigma$ AND $\sqcup \in \Gamma$

q_0 = Initial State $\quad q_0 \in Q$

$q_{ACCEPT} \in Q$
$q_{REJECT} \in Q$

WE ONLY NEED ONE ACCEPT AND ONE REJECT STATE.

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$

Transition function

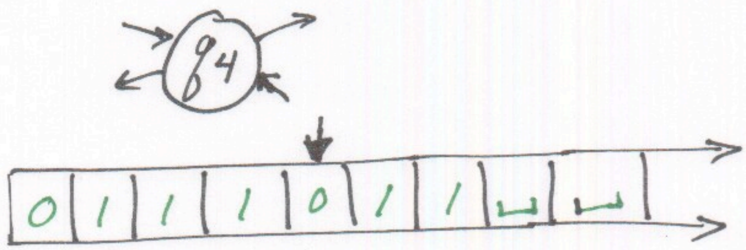
"CONFIGURATION"

GIVES THE ENTIRE STATE OF THE MACHINE

SNAPSHOT OF EXECUTION AT SOME STEP.

NEED:

- CONTENTS OF THE TAPE.
- LOCATION OF THE "TAPE HEAD"
- CURRENT STATE.



A CONFIGURATION IS A STRING LIKE THIS:

$$0\ 1\ 1\ 1\ q_4\ 0\ 1\ 1$$

A SEQUENCE OF CONFIGURATIONS, STARTING WITH THE "START CONFIGURATION", AND ENDING WITH AN [ACCEPTING]* CONFIGURATION, AND CONTAINING ONLY LEGAL TRANSITIONS PROVIDES A

"COMPUTATION HISTORY"

* OR "REJECTING"

DECIDABLE LANGUAGES

ALSO:
"RECURSIVE"
"COMPUTABLE"
"SOLVABLE"

When given a string as input, the TM will always halt.
The TM will ACCEPT if it is in L.
The TM will REJECT if it is not in L.

TURING RECOGNIZABLE LANGUAGES

ALSO:
"RECURSIVELY ENUMERABLE," **RE**
"PARTIALLY DECIDABLE"
"SEMI-DECIDABLE"

When given a string that is in the language, the TM will always HALT and ACCEPT. When given a string that is not in the Language, the TM will either REJECT or LOOP.

NOT TURING RECOGNIZABLE LANGUAGES

CAN'T EVEN RECOGNIZE MEMBERS RELIABLY!

ALSO:
NOT RECURSIVELY ENUMERABLE
NOT R.E.
NOT PARTIALLY DECIDABLE

TURING MACHINE USES

* TO "DECIDE" A LANGUAGE
* TO "RECOGNIZE" A LANGUAGE
* TO COMPUTE A FUNCTION

"COMPUTABLE"

≡ DECIDABLE
"TOTALLY COMPUTABLE"
DEFINED ON ALL INPUTS.

PARTIALLY COMPUTABLE FUNCTIONS

UNDEFINED ON SOME INPUTS
"SEMI-DECIDABLE FUNCTIONS"

THE CHURCH-TURING THESIS

1930's: What does "COMPUTABLE" mean?

ALONZO CHURCH: LAMBDA CALCULUS

ALAN TURING: TMs.

Several variations on TURING MACHINES.

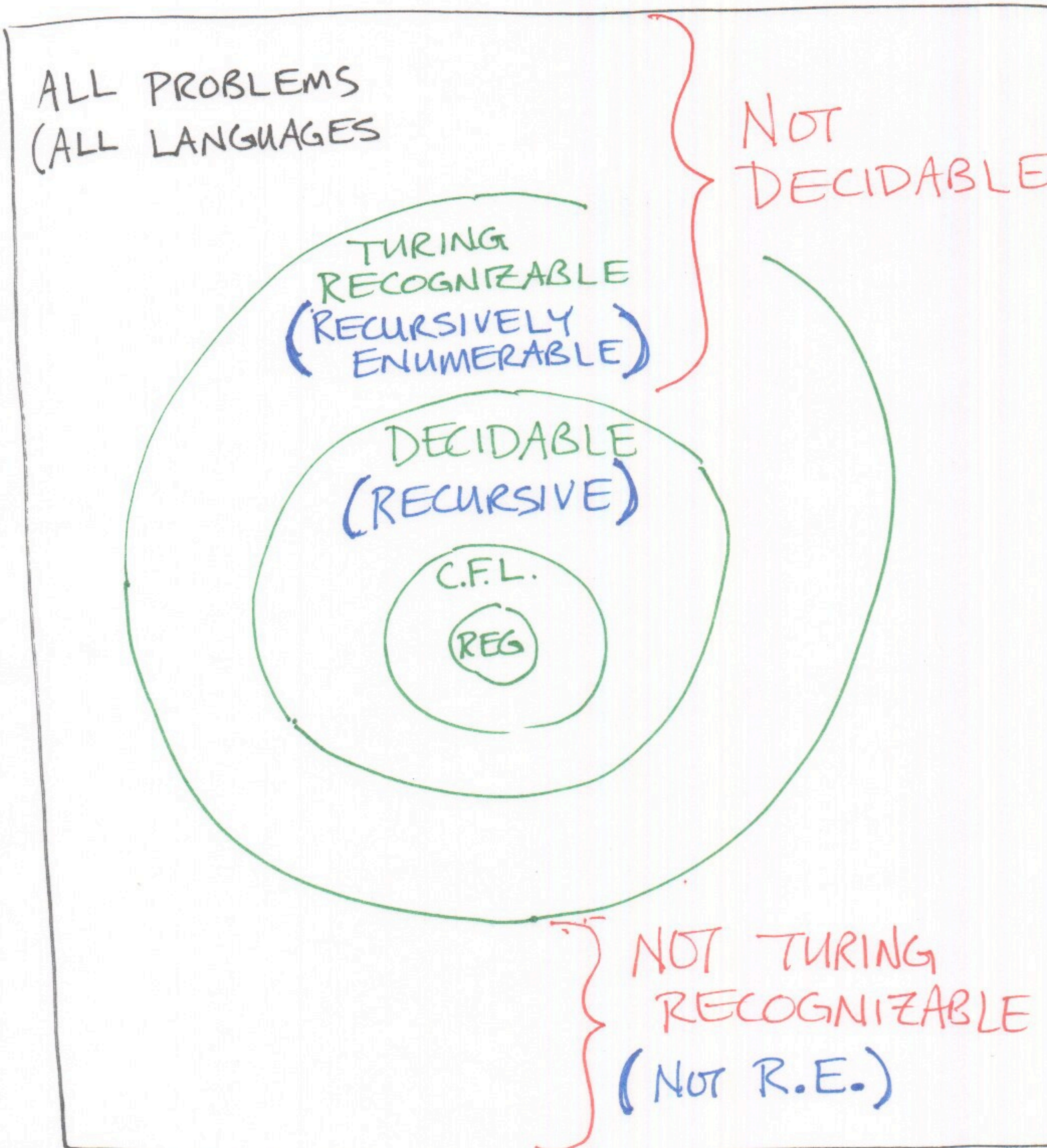
- ONE TAPE OR MANY?
- INFINITE ON BOTH ENDS?
- TINY ALPHABET $\{0, 1\}$ OR NOT?
- CAN THE HEAD ALSO STAY IN THE SAME PLACE?
- ALLOW NONDETERMINISM.

All variations are equivalent in computing capability!

TM's AND LAMBDA CALCULUS are also equivalent in power.

CONCLUSION: (OR DEFINITION?)

"Algorithmically computable"
EQUALS
"Computable by a TM."

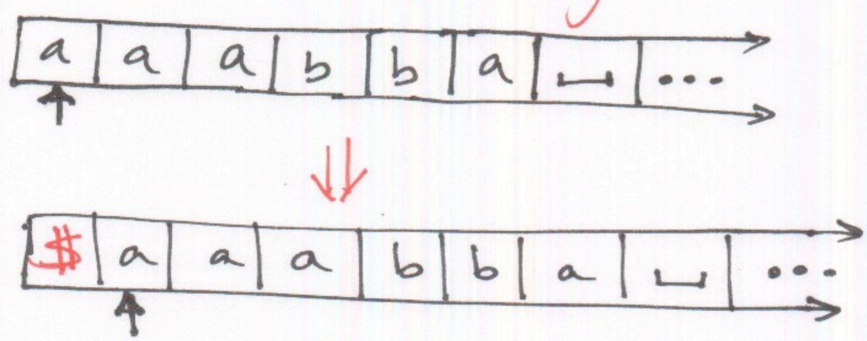


A VENN DIAGRAM

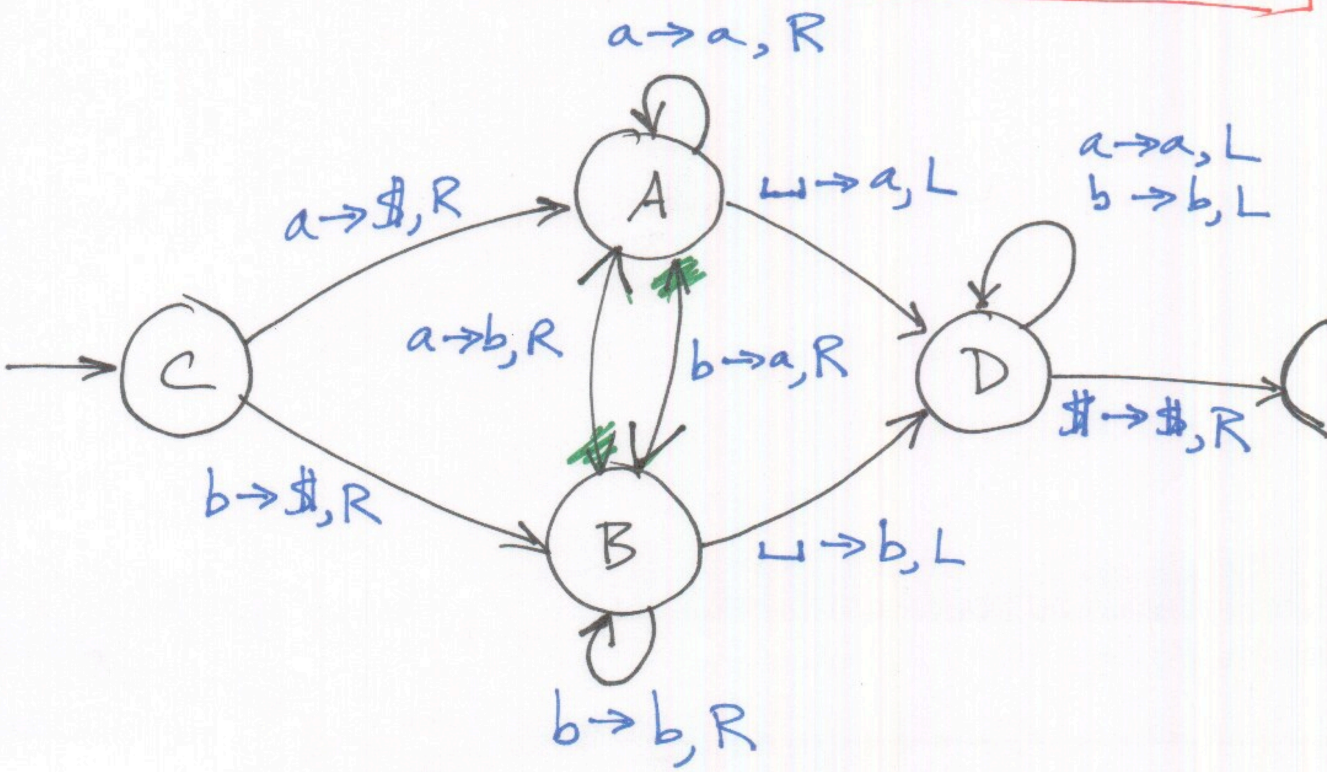
PROBLEM

How can we recognize the left end of the tape?

<u>GOAL</u>: Want to put a special symbol \$ on the left end and shift the input over 1 cell to the right.



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



Q: How much TM programming do you need to do?

A: Just enough to get the idea and to convince yourself that all programs/algorithms can be implemented on a TM.

Machine Code

0110, 1100

⇓

Assembly Code

ADD R1, R2, R3

⇓

C code

i = (2+k)*n;

⇓

Algorithms

If $S \cap T = \emptyset \ldots$

No implementation details.

TURING MACHINES

STATES, TRANSITION FUNCTION

(complete TM specification)

⇓

Outline of Algorithm

Still talking about Tape Head movement, Data representation

⇓

High-level specification of algorithm

No TM-specific details

If $S \cap T = \emptyset \ldots$

EXAMPLE

Build a TM to recognize the language $0^N 1^N 0^N$.

"Build a TM to "**decide**" the language."

This language is <u>not</u> context-free. So this will prove

CONTEXT-FREE LANGUAGES $\subset$ DECIDABLE LANGUAGES

proper subset

We already have a TM to turn $0^N 1^N$ into $X^N Y^N$ and to ~~recognize~~ decide that language.

IDEA

Use that TM as a SUBROUTINE!

STEP 1:

000000 111111 000000

$\Downarrow$

XXXXXX YYYYYY 000000

Reject if problems.

STEP 2:

Build a similar TM to recognize $Y^N 0^N$

STEP 3:

Build the final TM by "gluing" these smaller TM's together into one larger TM.

PROBLEM:

Compare two strings.

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

SOLUTION:

Use a new symbol, such as "x".

Turn each symbol into an x after it has been examined.

$$a\,a\,b\,a\,c\,\#\,a\,a\,b\,a\,c$$

$$\Downarrow$$

$$x\,a\,b\,a\,c\,\#\,x\,a\,b\,a\,c$$

$$\Downarrow$$

$$x\,x\,b\,a\,c\,\#\,x\,x\,b\,a\,c$$

$$\Downarrow^*$$

$$x\,x\,x\,x\,x\,\#\,x\,x\,x\,x\,x$$

PROBLEM:

Do it nondestructively, without losing the original strings.
(Perhaps this task is part of a larger task.)

SOLUTION:

"MARK" each symbol to keep track of what we've already done.
Add some new symbols to help.

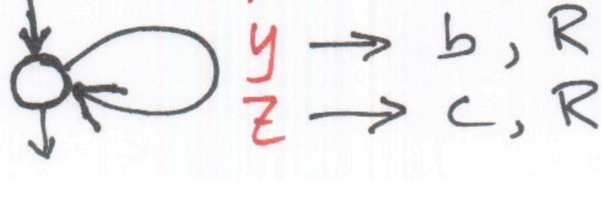
$a \rightarrow x$
$b \rightarrow y$
$c \rightarrow z$

a a b c a # a a b c a

$\Downarrow$

x x y z a # x x y z a

Later, restore the strings, if we need to:

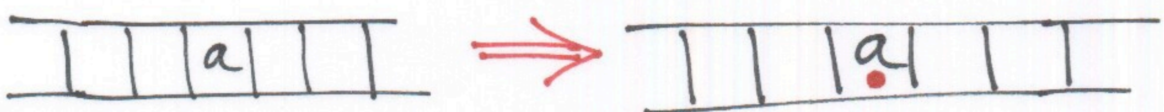


Programming Technique: Marking Symbols

"Mark each symbol with a dot."

$$\Gamma = \{ a, b, c, \underset{\bullet}{a}, \underset{\bullet}{b}, \underset{\bullet}{c}, \# \}$$

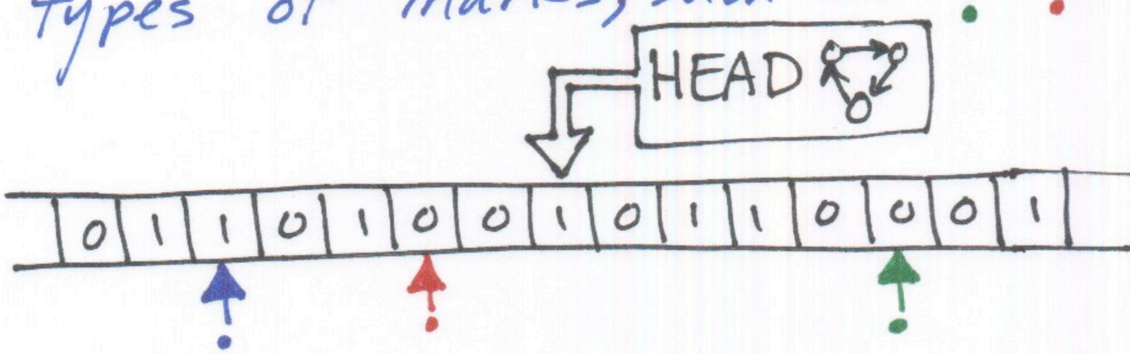
"Put a dot under this symbol."



"Remember this location"

$\Rightarrow$ Mark that symbol with a dot.

We could have several different types of marks, such as



"Let P point to the beginning of the second string and Let q point to..."

THEOREM

EVERY MULTITAPE TURING MACHINE HAS AN EQUIVALENT SINGLE-TAPE TURING MACHINE.

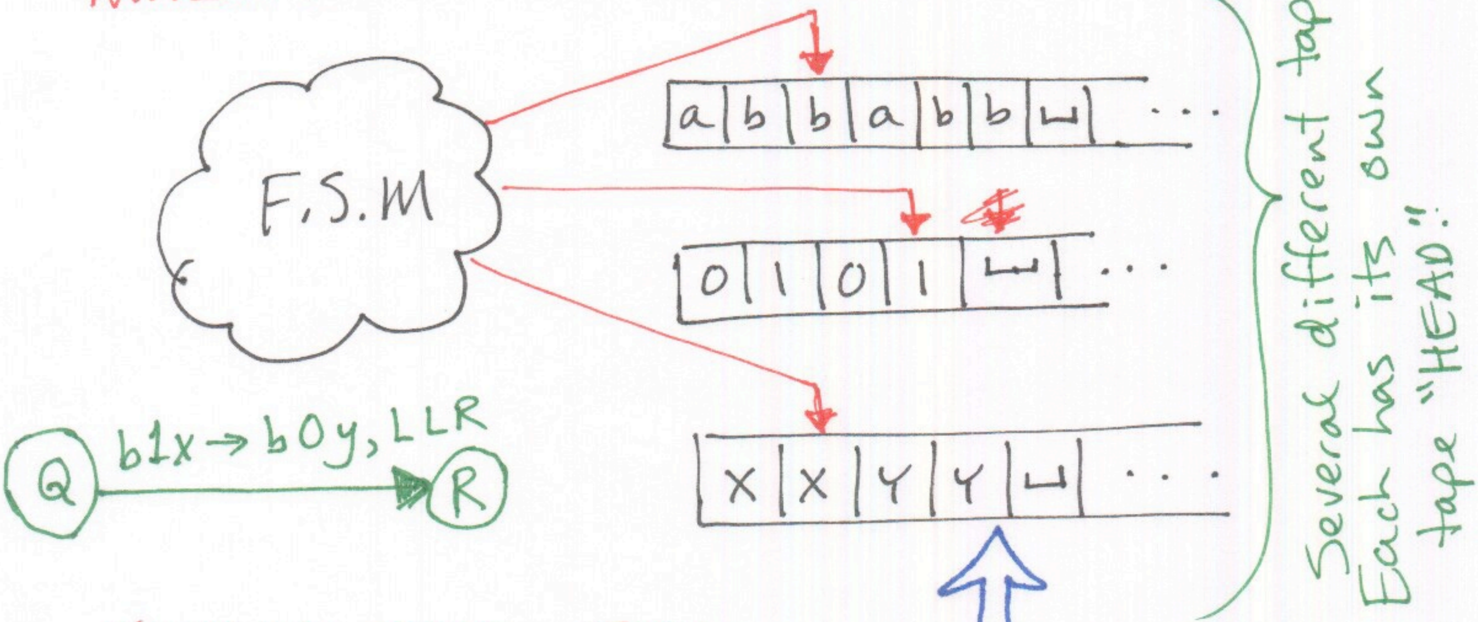
"EQUIVALENT" MEANS IT DECIDES/RECOGNIZES THE SAME LANGUAGES. IT'S NOT ABOUT SPEED, EFFICIENCY OR EASE OF PROGRAMMING.

PROOF

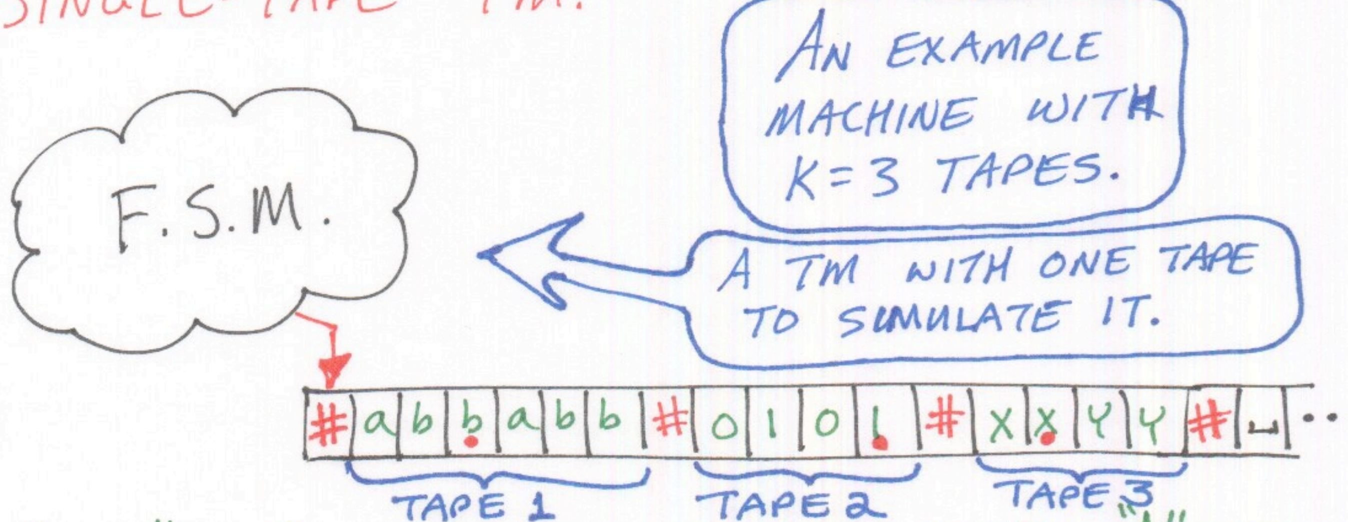
GIVEN A MULTITAPE TM, SHOW HOW TO BUILD A SINGLE-TAPE TM.

- 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 MORE MOVES IN THE SINGLE-TAPE TM.

MULTITAPE TM:



SINGLE-TAPE TM:



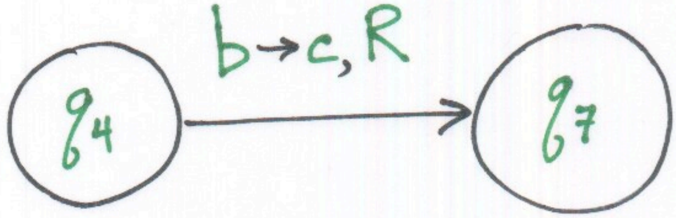
- ADD "DOTS" TO SHOW WHERE HEAD "K" IS.
- To simulate a transition from state Q, we must scan our tape to see which symbols are "UNDER" the K tape heads.
- Once we determine this, and are ready to "MAKE" the transition, we must scan across the tape again to update the cells and move the dots.
- Whenever one head moves off the right end, we must shift our tape so we can insert a ⊔.

NONDETERMINISTIC TURING MACHINES

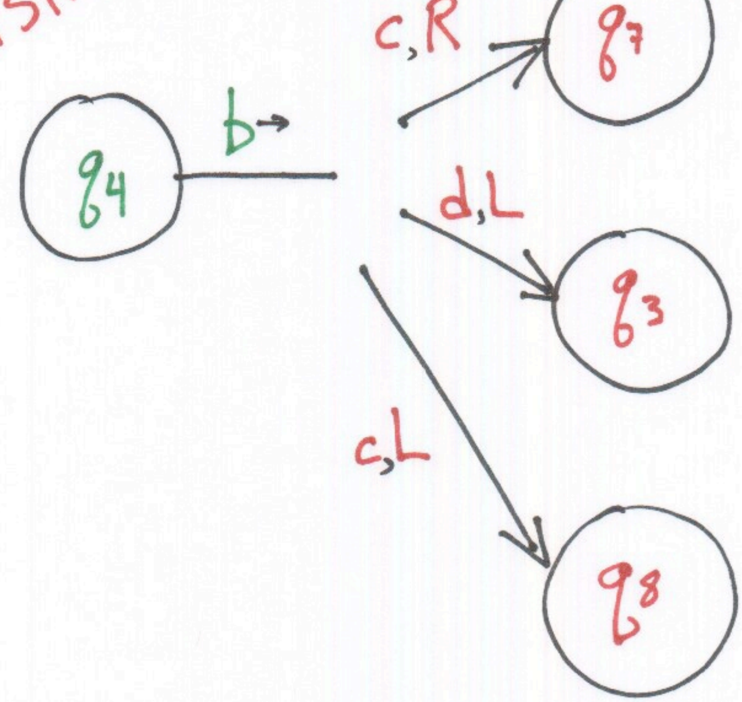
TRANSITION FUNCTION

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

Deterministic



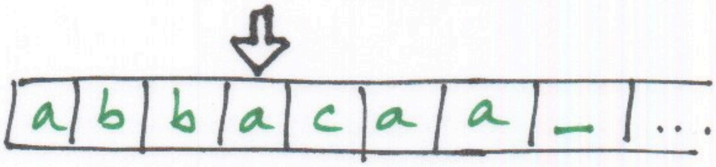
Nondeterministic



A "CONFIGURATION" is...

- A way to represent the entire state of a TM at one moment during a computation.
- A string which captures

  THE CURRENT STATE
  THE CURRENT POSITION OF HEAD
  THE ENTIRE TAPE CONTENTS.

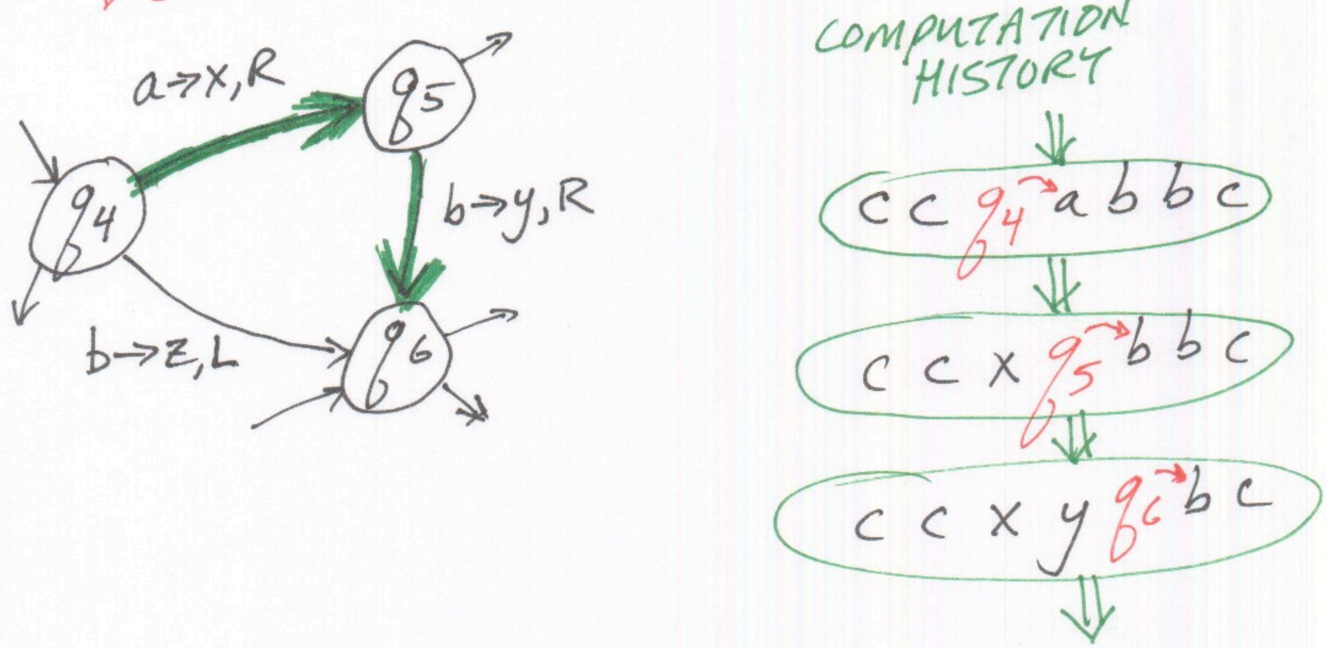


a b b q_{37} a c a a

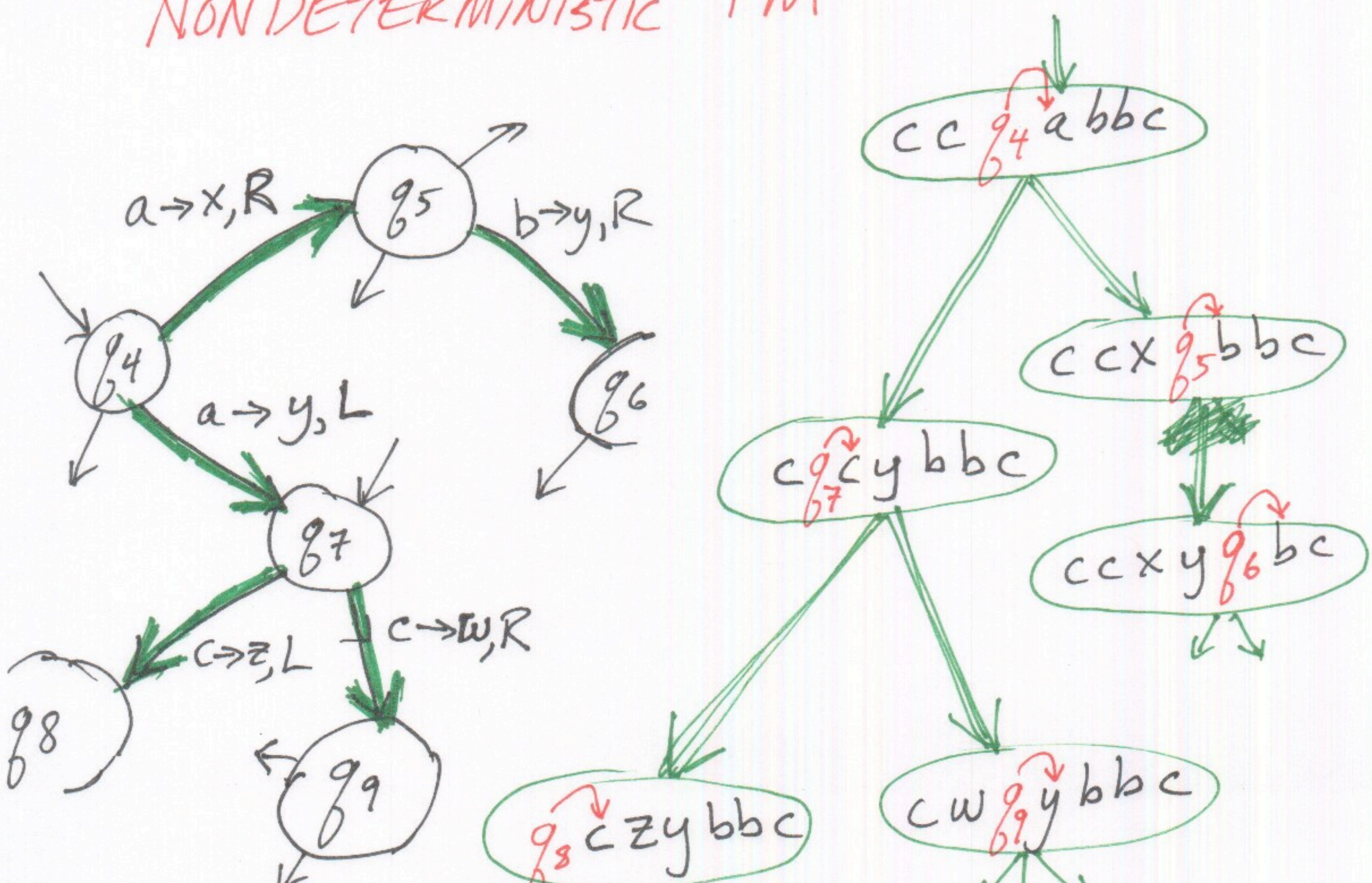
WITH NONDETERMINISM:

At each moment in the computation there can be <u>MORE THAN ONE</u> successor configuration.

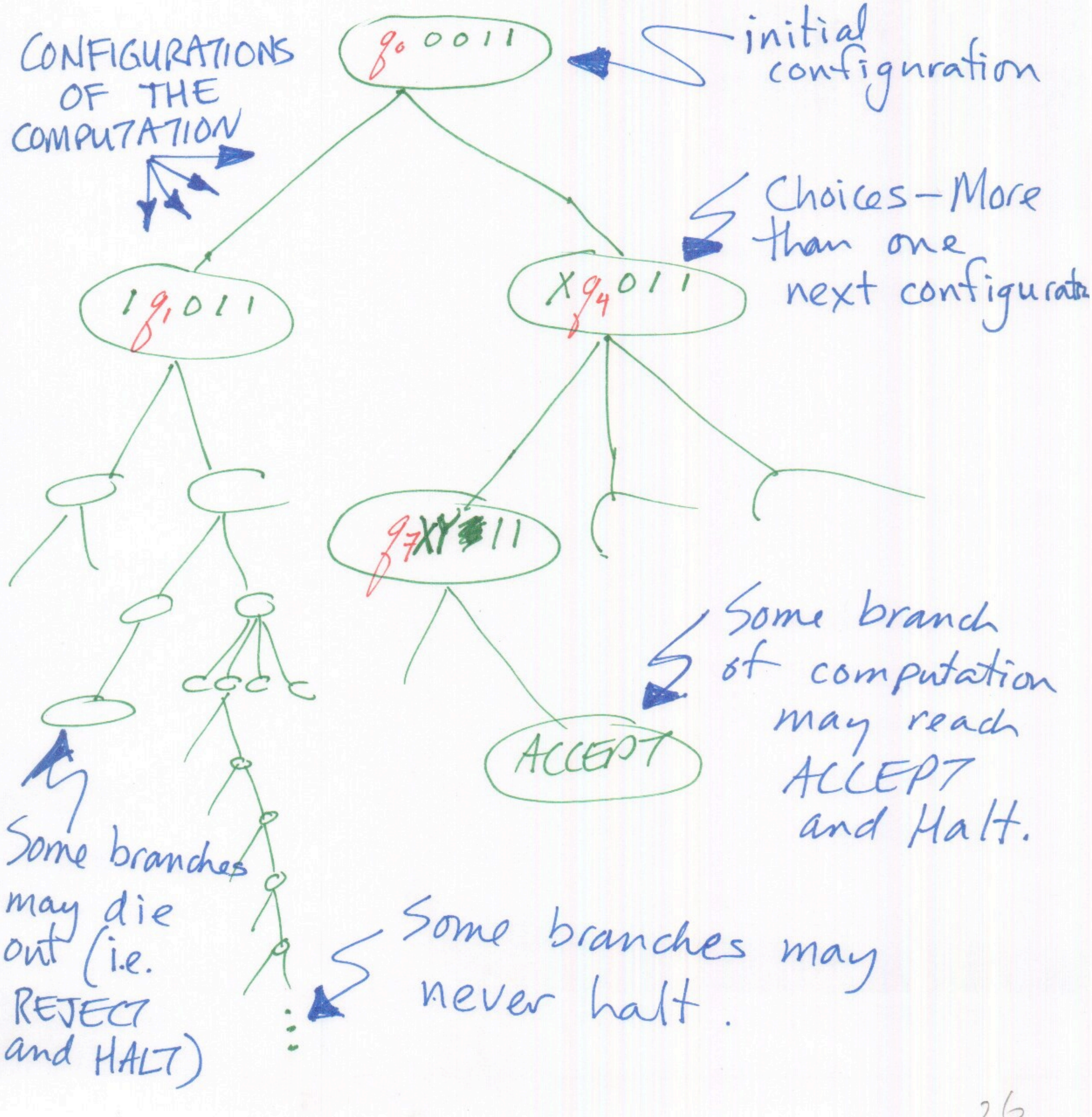
DETERMINISTIC TM



NONDETERMINISTIC TM



A TREE SHOWS THE COMPUTATION OF A NON-DETERMINISTIC TM.



OUTCOMES OF A NONDETERMINISTIC COMPUTATION:

ACCEPT

IF <u>ANY</u> BRANCH OF THE COMPUTATION ACCEPTS, THEN THE NONDETERMINISTIC TM WILL ACCEPT.

REJECT

IF <u>ALL</u> BRANCHES OF THE COMPUTATION HALT AND REJECT (i.e., NO BRANCHES ACCEPT, BUT ALL COMPUTATION HALTS), THEN THE NONDETERMINISTIC TM REJECTS.

LOOP

COMPUTATION CONTINUES, BUT "ACCEPT" IS NEVER ENCOUNTERED.

SOME BRANCHES IN THE COMPUTATION HISTORY ARE INFINITE.

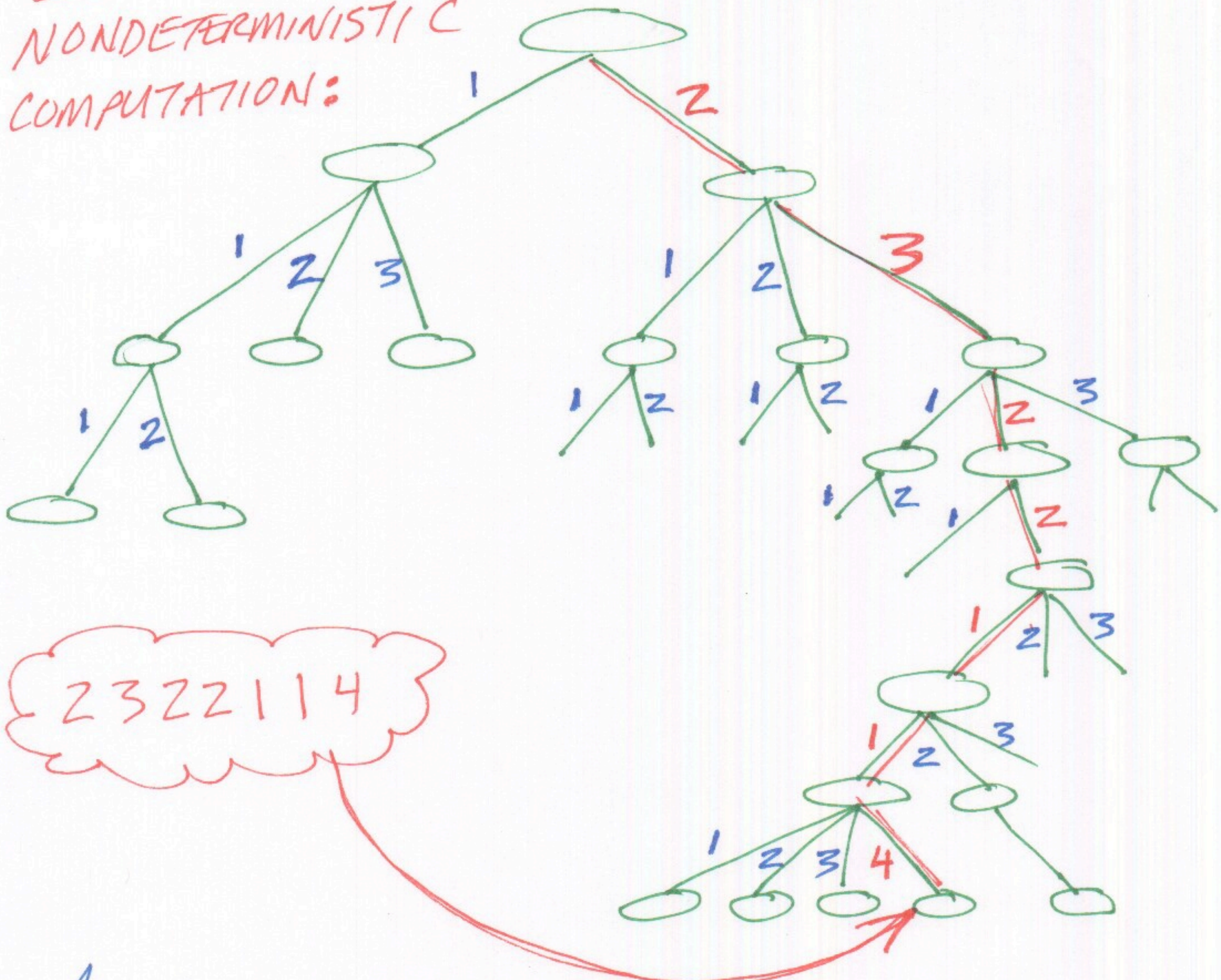
THEOREM

EVERY NONDETERMINISTIC TM HAS AN EQUIVALENT DETERMINISTIC TM.

PROOF

- GIVEN A NON-DETERMINISTIC TM, **(N)** SHOW HOW TO CONSTRUCT AN EQUIVALENT DETERMINISTIC TM **(D)**
- IF **N** ACCEPTS (ON ANY BRANCH) THEN **D** WILL ACCEPT.
- IF **N** HALTS ON EVERY BRANCH WITHOUT ANY "ACCEPTS", THEN **D** WILL HALT AND REJECT.
- APPROACH: SIMULATE **N**; SIMULATE <u>ALL</u> BRANCHES OF COMPUTATION; SEARCH FOR <u>ANY</u> WAY **N** CAN ACCEPT.

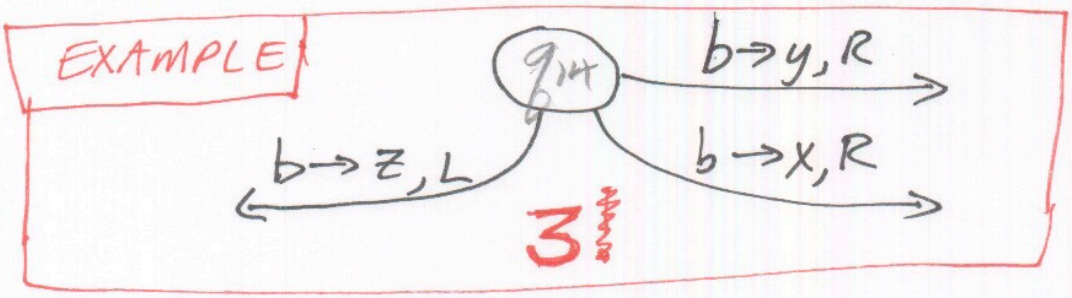
THE "COMPUTATION HISTORY" IS A TREE SHOWING <u>ALL</u> POSSIBLE BRANCHES/CHOICES IN A NONDETERMINISTIC COMPUTATION:

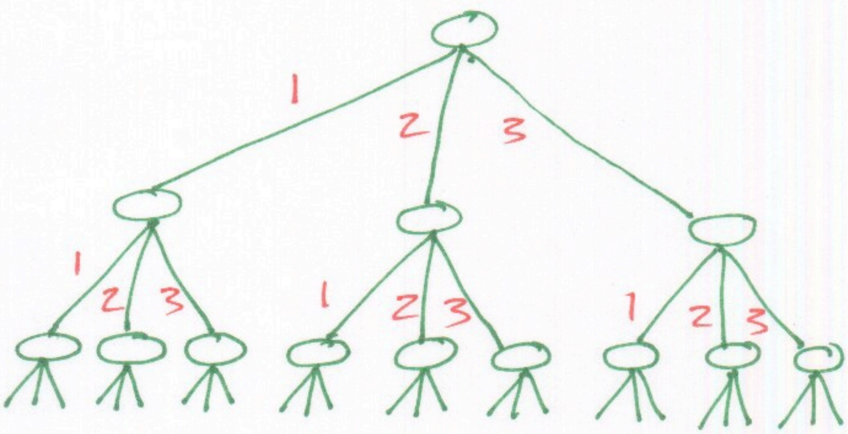


- A PATH TO ANY NODE IS GIVEN BY A NUMBER.
- SEARCH THE TREE, LOOKING FOR "ACCEPT."
- SEARCH ORDER?
  - DEPTH-FIRST? ← NO!
  - BREADTH-FIRST! ← YES!
- TO EXAMINE A NODE:
  - PERFORM THE <u>ENTIRE</u> COMPUTATION FROM SCRATCH.
  - THE PATH NUMBERS TELL WHICH OF THE MANY NONDETERMINISTIC CHOICES TO MAKE

HOW MANY CHOICES AT EACH STEP IN THE COMPUTATION?

EXAMINE THE NONDETERMINISTIC MACHINE; THERE WILL BE SOME MAXIMUM. #.





BREADT-FIRST SEARCH ORDER:

ε

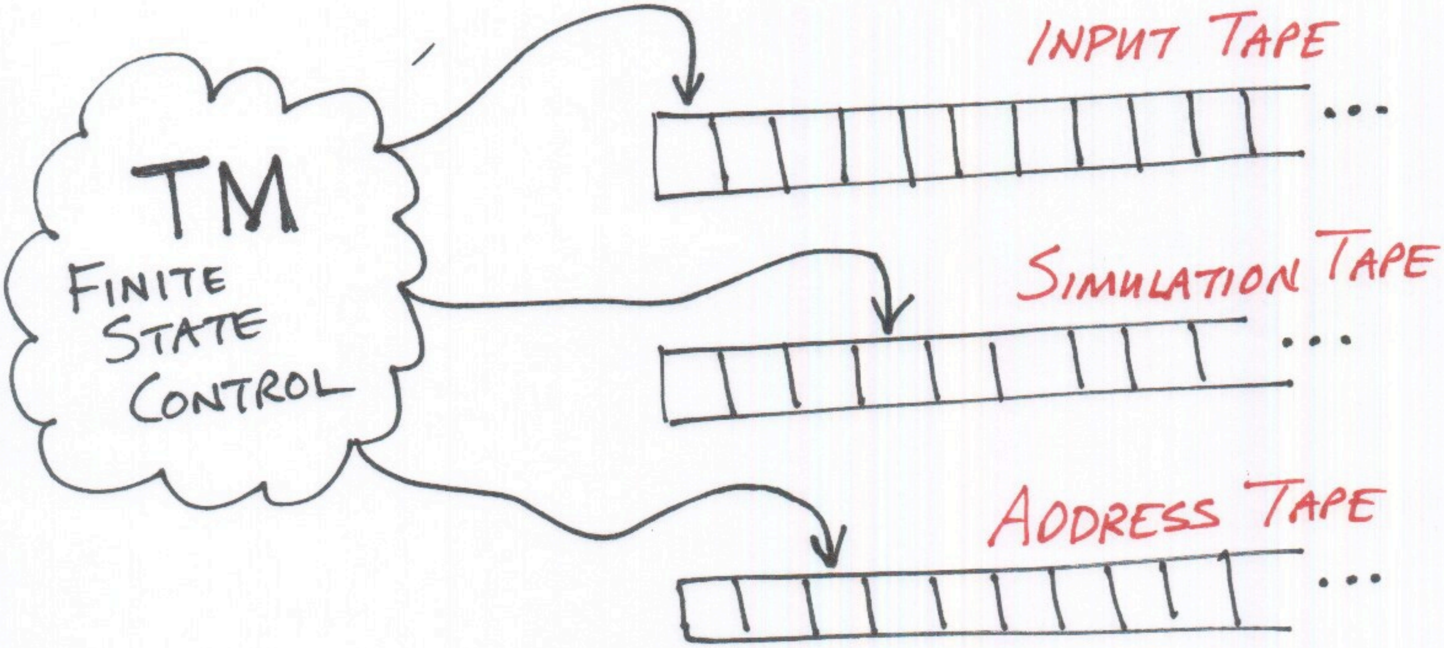
1
2
3

1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3

1 1 1
1 1 2
1 1 3
1 2 1
1 2 2
1 2 3
⋮

IN ANY PARTICULAR COMPUTATION THERE WILL BE FEWER THAN 3 CHOICES AT MOST OF THE COMPUTATION STEPS.

SOME POINTS WILL HAVE ZERO CHOICES ⇒ THESE BRANCHES OF THE NONDETERMINISM HALT AND REJECT.



INPUT TAPE

Initial input; never modified.

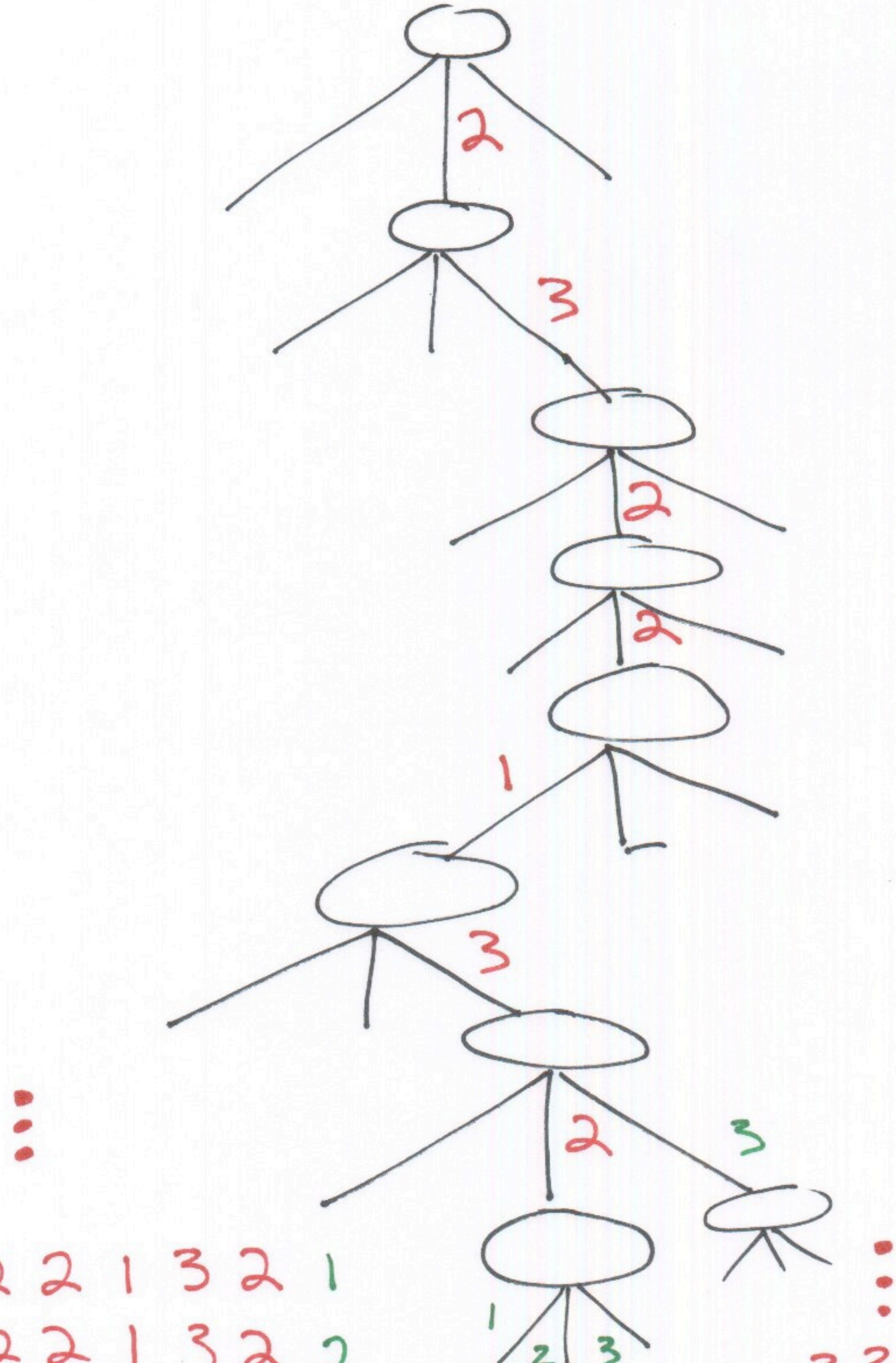
SIMULATION TAPE

USED LIKE THE TAPE OF A DETERMINISTIC TM TO PERFORM THE SIMULATION.

ADDRESS TAPE

Used to control the breadth-first Search.

Tells which choices to make during a simulation.



2 3 2 2 1 3 2 1
2 3 2 2 1 3 2 2
2 3 2 2 1 3 2 3
2 3 2 2 1 3 ~~2~~3 1
2 3 2 2 1 3 ~~2~~3 2
2 3 2 2 1 3 ~~2~~3 3

3 3 1
⋮ 3 3 2
3 3 3 3 3 3 3
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 2
1 1 1 1 1 1 1 3
1 1 1 1 1 1 2 1

32.1

ALGORITHM

INITIALLY: TAPE 1 CONTAINS THE INPUT.
TAPES 2 AND 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 OUT.")

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."

A LANGUAGE IS "TURING RECOGNIZABLE" IFF SOME NON-DETERMINISTIC TURING MACHINE RECOGNIZES IT.

HALTING?

If a nondeterministic TM halts on ALL branches without ACCEPTING, then it REJECTS

A LANGUAGE IS "DECIDABLE" IFF SOME NON-DETERMINISTIC TURING MACHINE DECIDES IT.

DECIDES?

- Will always halt!
- Will always ACCEPT or REJECT!
- Will never LOOP!

ANY ARBITRARY PROBLEM CAN BE EXPRESSED AS A LANGUAGE.

Any instance of the "problem" is ENCODED into a string.

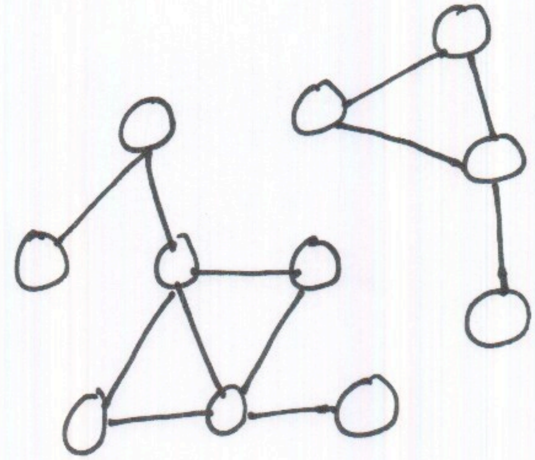
THE STRING IS IN THE LANGUAGE $\equiv$ THE ANSWER IS "YES"

THE STRING IS NOT IN THE LANGUAGE $\equiv$ THE ANSWER IS "NO"

EXAMPLE

UNDIRECTED GRAPHS

IS THIS GRAPH "CONNECTED"?



WE MUST ENCODE THE PROBLEM INTO A LANGUAGE.

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

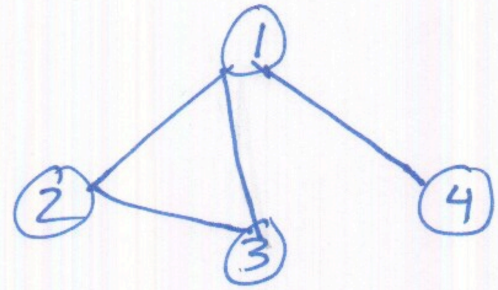
We would like to find a TM to DECIDE this language.

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.

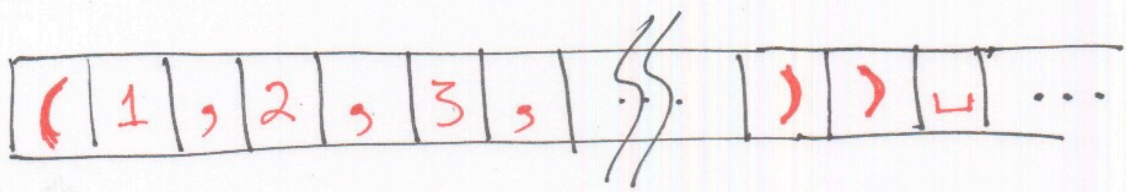
ONE REASONABLE REPRESENTATION:



$\langle G \rangle =$

$$\underbrace{(1,2,3,4)}_{\text{List of node ``names''}}\underbrace{\Big((1,2),(2,3),\underbrace{(1,3)}_{\text{An edge from (1) to (3)}},(1,4)\Big)}_{\text{LIST OF EDGES}}$$

$$\Sigma = \{\, (\,,\ )\,,\ ,\,,\ 1,\ 2,\ 3,\ 4,\ \cdots,\ 9 \,\}$$



REPRESENTING NUMBERS ON A TAPE

DECIMAL

$$\Sigma = \{0, 1, 2, \cdots, 9 \cdots, \#, \$, x \cdots\}$$

$$\cdots \; | \, , \, | \, 4 \, | \, 3 \, | \, 9 \, | \, , \, | \quad | \cdots$$

BINARY

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

$$\cdots \; | \, , \, | \underbrace{\, 1 \, | \, 0 \, | \, 1 \, | \, 1 \, | \, 0 \,}_{=22} | \, , \, | \cdots$$

UNARY

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

$$\cdots \; | \, , \, | \underbrace{\, 1 \, | \, 1 \, | \, 1 \, | \, 1 \, | \, 1 \, | \, 1 \, | \, 1 \, | \, 1 \,}_{=8} | \, , \, | \cdots$$

(This is no more than a programming detail.)

ALGORITHM = TURING MACHINE

- HIGH-LEVEL SPECIFICATION
  - PSEUDO-CODE
  - EXPRESSED IN PGMMING LANGUAGE
- IMPLEMENTATION-LEVEL
  - CONTENTS OF THE TAPE.
  - DATA REPRESENTATION.
  - MOTION OF THE TAPE HEAD.
  - MORE DETAIL, BUT STILL ABSTRACT
- T.M. SPECIFICATION
  - STATES
  - ALPHABETS
  - TRANSITION FUNCTION
  - FULLY DETAILED (& INCOMPREHENSIBLE?)

Once we are comfortable with T.M. specification details, we'll start to give more abstract algorithms, with the understanding that we could build the exact T.M. whenever necessary.

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 DIGIT, ...
    - CHECK THAT ALL NODES ARE DIFFERENT, i.e., NO REPEATS.
  - CHECK EDGE LIST...

etc.

- MARK FIRST NODE.
  - PLACE A DOT UNDER THE FIRST NODE IN NODE LIST.
- SCAN THE NODE LIST TO FIND A NODE THAT IS NOT MARKED..

etc., etc.

ENUMERATORS

LIKE A TURING MACHINE:

INFINITE TAPE

FINITE STATE CONTROL

PLUS....

A PRINTER

OPERATION

- THE TAPE IS INITIALLY EMPTY (i.e., NO INPUT).
- THE PRINTER PRODUCES A SERIES OF STRINGS.
- THE MACHINE ENUMERATES (i.e., IT "LISTS OUT"/"PRINTS") THE STRINGS IN A LANGUAGE.
- IT MAY HALT OR LOOP.
- THE LANGUAGE MAY BE INFINITE.
- IT MAY PRINT OUT DUPLICATES (JUST IGNORE DUPLICATE STRINGS).
- IT MAY PRINT IN ANY ORDER

THEOREM

A LANGUAGE IS TURING-RECOGNIZABLE IFF SOME ENUMERATOR ENUMERATES IT.

PROOF

GIVEN "E" CONSTRUCT A T.M "M".

ON INPUT w...
RUN "E"
COMPARE EACH OUTPUT STRING TO w.
IF WE FIND A MATCH, ACCEPT.

GIVEN A TM "M", CONSTRUCT AN ENUMERATOR "E".

CONSTRUCT E, USING M AS A "SUBROUTINE".

RUN M ON <u>ALL</u> POSSIBLE STRINGS in Σ^*.

IF M EVER ACCEPTS A STRING, THEN PRINT IT OUT.

PROBLEM?

M MIGHT LOOP ON SOME PARTICULAR STRINGS.

WE MUST RUN ALL THESE SIMULATIONS IN PARALLEL!

GOAL

Run a TM on all strings in Σ^* simultaneously (i.e., "in parallel")

APPROACH

We can list out all strings in

$$\Sigma^* = \{ s_1, s_2, s_3, \cdots \}$$

EXAMPLE: ϵ, 0, 1, 00, 01, 10, 11, 000, ...

The computation on any string s_i may be infinite!

We must not get stuck on some string.

EXAMPLE: S_4 might infinite loop, but S_7 might ACCEPT

∴ INTERLEAVE THE COMPUTATIONS.

Work on S_1 a little bit.

Work on S_2 a little bit.

⋮

Eventually, we must go back to s_1 and do a little more work.

ALGORITHM

FOR $i = 1, 2, 3, \cdots$ (infinite loop)

FOR $j = 1$ TO i

Simulate M, the Turing Machine.

Use s_j as input.

Run simulation for i steps.

If M accepts s_j

(within i steps)

Then PRINT s_j

END

END

Note: Every (i, j) pair will eventually be encountered.

EXAMPLE

ASSUME

S_1 is ACCEPTED after 3 steps.

S_3 is ACCEPTED after 4 steps.

	1	2	3	4	5	6	•••
S_1			A	A	A	A	...
S_2							
S_3				A	A	A	...
S_4							
S_5							
S_6							

Output:

S_1

S_1

S_3

S_1

S_3

⋮