CHAPTER 3: TURING MACHINES

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

A NEW MODEL OF COMPUTATION

NOT MUCH MORE ELABORATE.

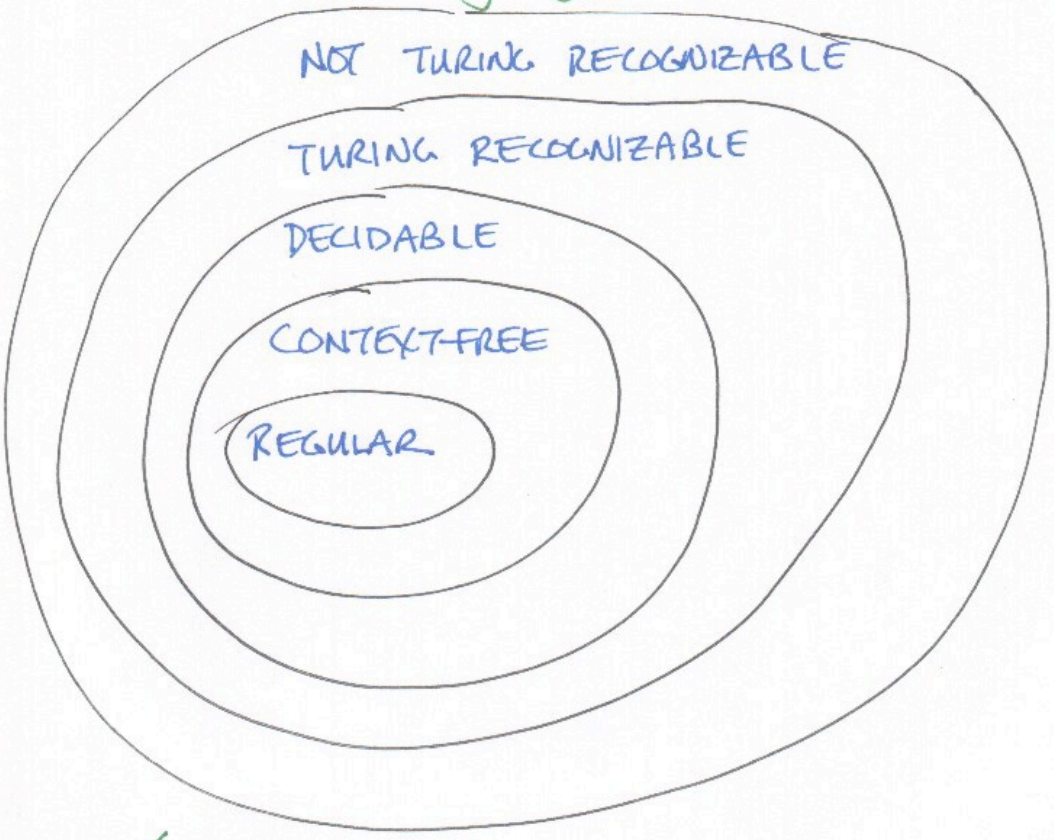
A "MODEL" FOR ALL COMPUTERS.

THREE NEW CLASSES OF LANGUAGE

- "DECIDABLE"
- "TURING RECOGNIZABLE"
- "NOT TURING RECOGNIZABLE"

CLASSES OF LANGUAGES

(The Language Onion)



VENN DIAGRAM

The relationship is proper subset.

TURING MACHINE DEFINITION

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

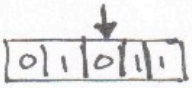
ALL VARIATIONS ARE EQUIVALENT!

WE'LL DISCUSS THIS LATER.

DATA STRUCTURE:

FSM

- THE INPUT STRING

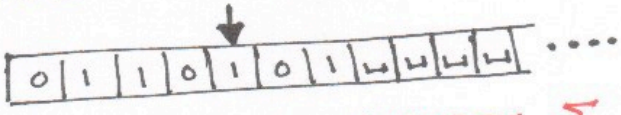


PDA

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

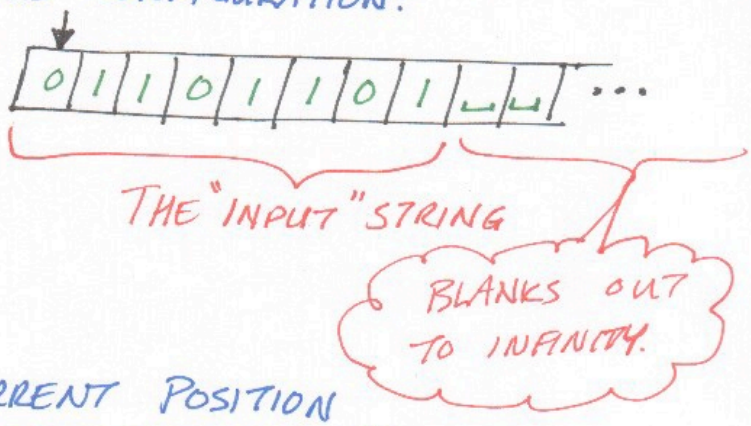
BUT ALSO COMMON:

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

THE "BLANK" SYMBOL IS SPECIAL

$$\sqcup \notin \Sigma$$

INITIAL CONFIGURATION:



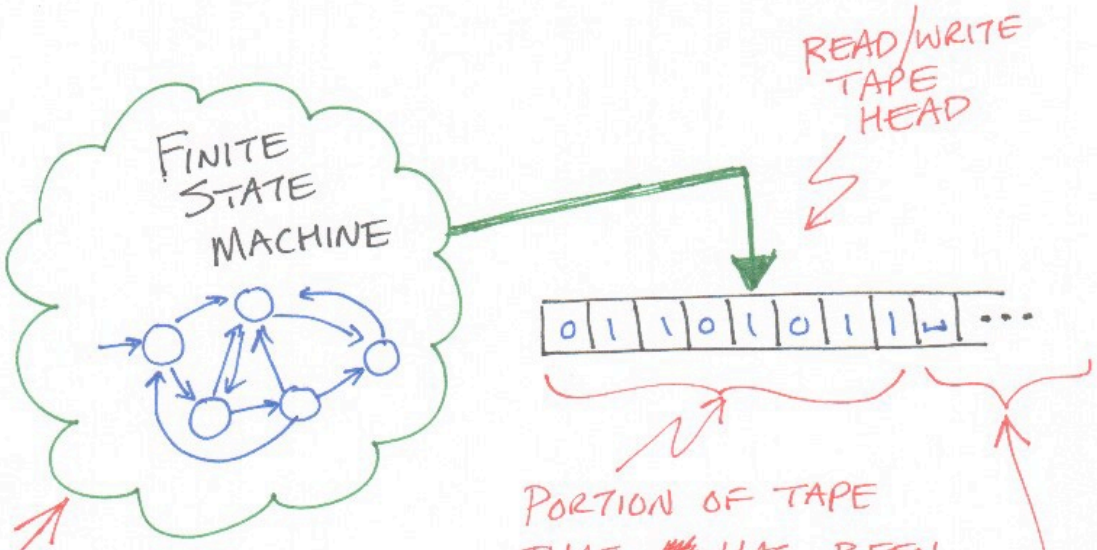
THE CURRENT POSITION ("THE TAPE HEAD")

INITIALLY AT THE LEFTMOST CELL.

CAN MOVE LEFT OR RIGHT.

CAN READ ("SCAN") THE CURRENT SYMBOL

CAN WRITE THE CURRENT SYMBOL



THE "CONTROL" PORTION; SIMILAR TO A P.D.A.; THE "PROGRAM"; DETERMINISTIC.

PORTION OF TAPE THAT HAS BEEN USED SO FAR.

UNUSED PORTION OF TAPE; INFINITE; FILLED WITH THE BLANK SYMBOL, ␣.

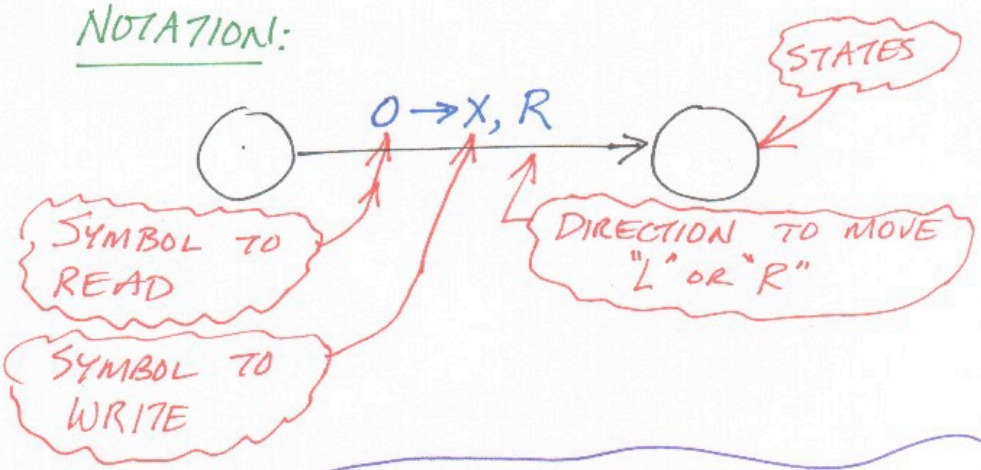
RULES OF OPERATION

AT EACH STEP OF THE COMPUTATION:

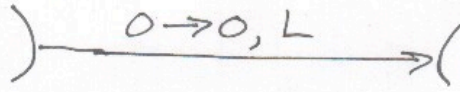
- READ THE CURRENT SYMBOL.
- UPDATE (i.e. WRITE) THE SAME CELL.
- MOVE EXACTLY ONE CELL EITHER LEFT OR RIGHT.

(IF AT LEFT END OF TAPE, WHEN MOVING LEFT, JUST STAY AT LEFT END.)

NOTATION:



DON'T WANT TO UPDATE THE CELL?
JUST WRITE THE SAME SYMBOL.



RULES OF OPERATION - 2

- CONTROL IS VIA A SORT OF FINITE STATE MACHINE.
- INITIAL STATE
- FINAL STATES:
  THE "ACCEPT" STATE
  THE "REJECT" STATE
- COMPUTATION CAN:

  HALT + "**ACCEPT**"

  The MACHINE ACCEPTS immediately when the ACCEPT STATE is reached.

  HALT AND "**REJECT**"

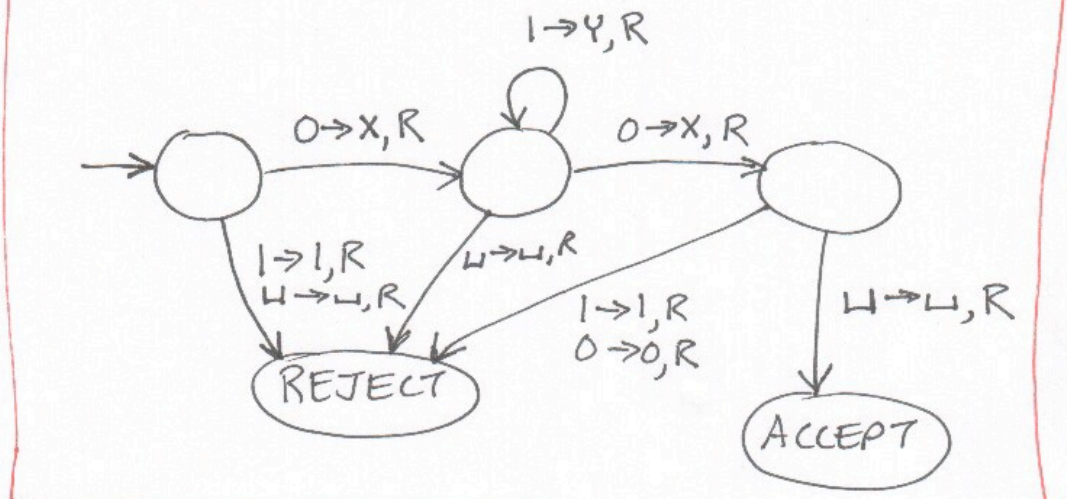
  The machine REJECTS immediately whenever the REJECT STATE is entered.

  "**LOOP**"

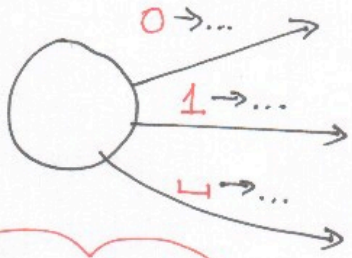
  The machine fails to halt.
- THE T.M. IS DETERMINISTIC.

An example

$L = 01^*0$



Is it deterministic?



Henceforth...

If an edge is missing...

Assume it leads to REJECT

DETAILS

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

Q = Set of states.

Σ = INPUT ALPHABET.

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

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

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.

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

Transition function

EXAMPLE

$L = 0^N 1^N$

INPUT ALPHABET:

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

ALGORITHM

CHANGE "0" TO "X"

MOVE RIGHT TO FIRST "1"

IF NONE: REJECT.

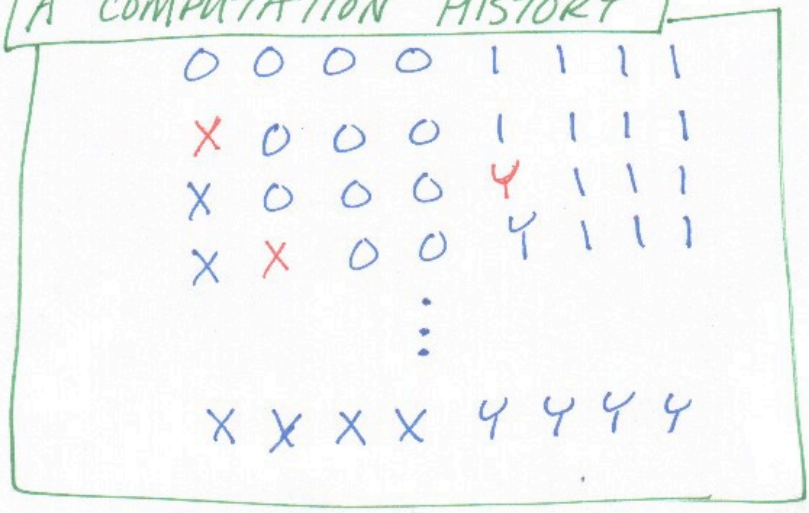
CHANGE "1" INTO "Y"

MOVE LEFT TO LEFTMOST "0"

REPEAT UNTIL NO MORE "0"S

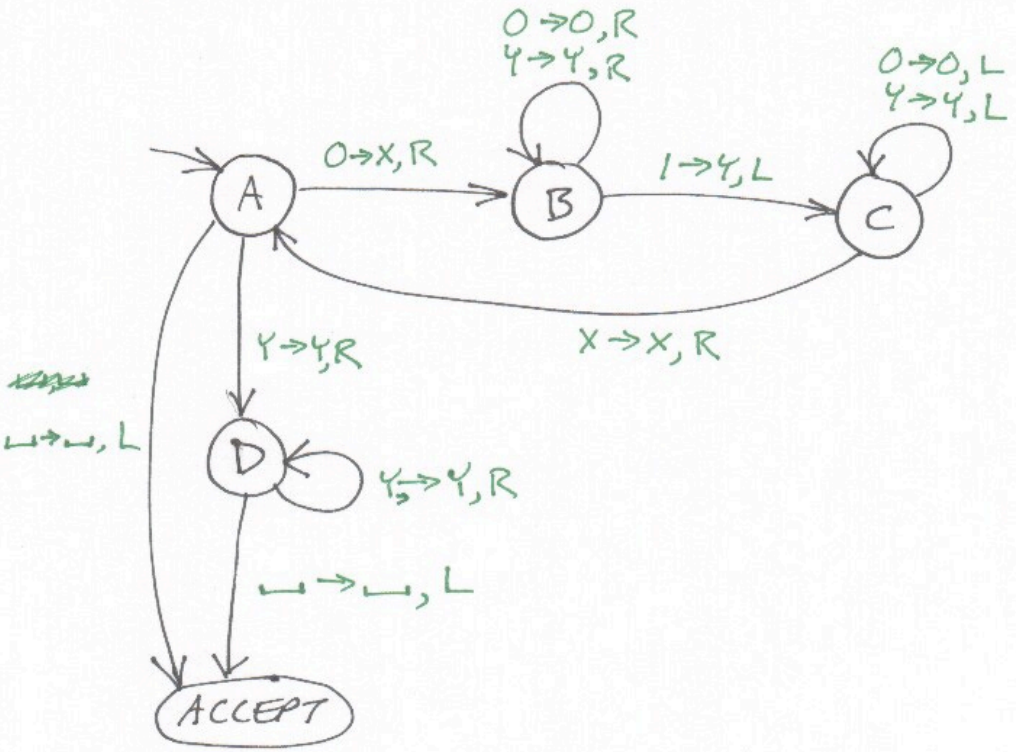
MAKE SURE NO MORE "1"S REMAIN

A COMPUTATION HISTORY



TAPE ALPHABET:

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



QUESTION:

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

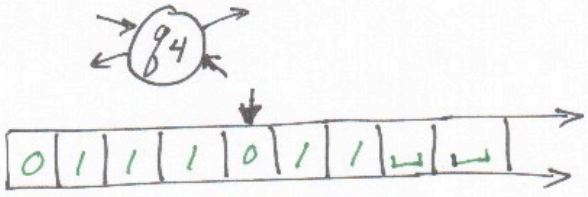
TM's model computers.
In this way they are similar!

"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 q4 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

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.

Also:
"Recursive"
"Computable"
"Solvable"

Turing Recognizable Languages

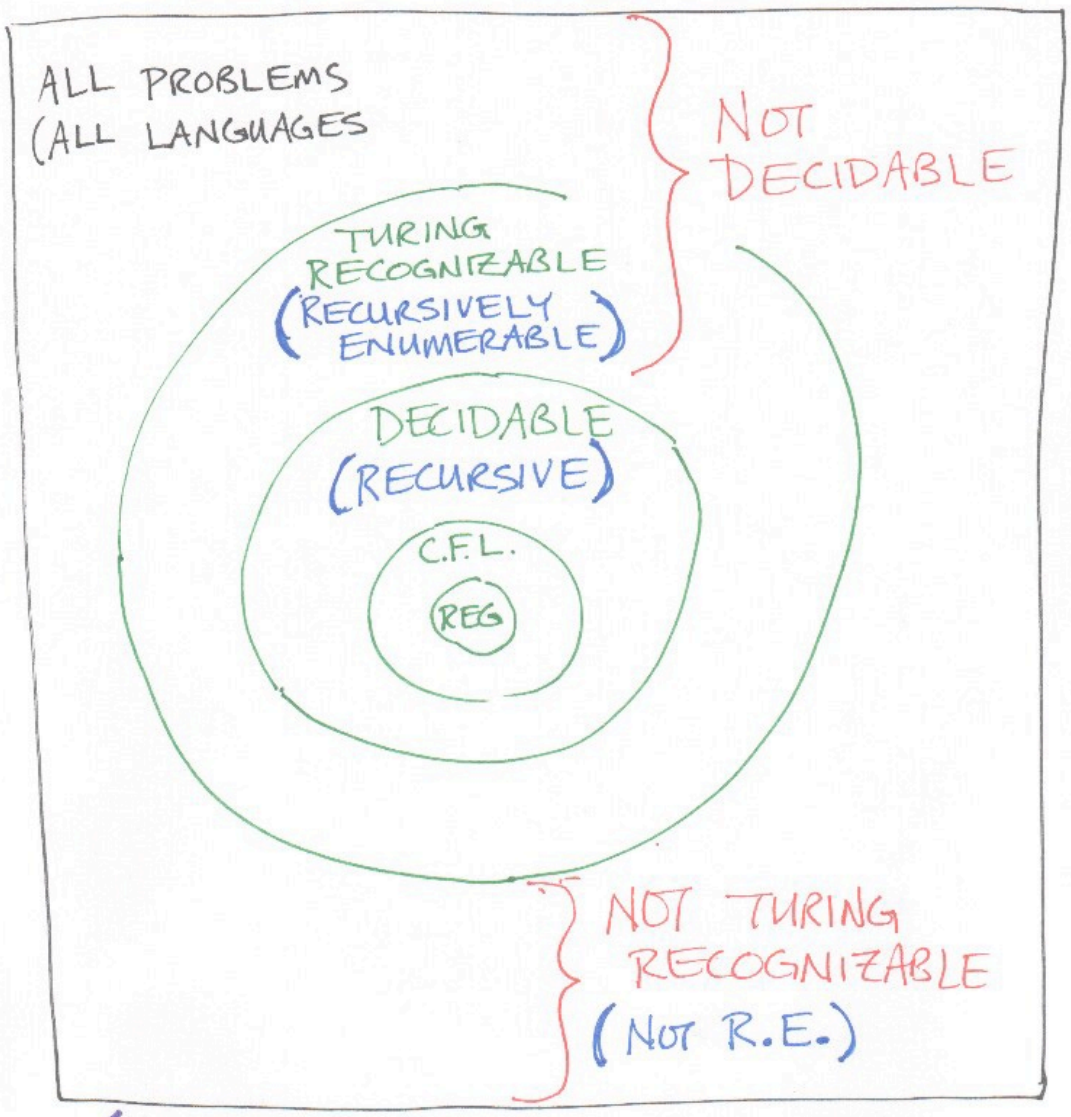
When given a string that is in the language, the TM will always HALT and ACCEPT. When given a string that is not in the Language, the TM will either REJECT or LOOP.

Also:
"Recursively Enumerable," RE
"Partially Decidable"
"Semi-Decidable"

Not Turing Recognizable Languages

Can't even recognize members reliably!

Also:
Not Recursively Enumerable
Not R.E.
Not Partially Decidable



A VENN DIAGRAM

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."

WARNING

"COMPUTABLE" ← Avoid this term

PARTIALLY COMPUTABLE

≡ TURING RECOGNIZABLE

MAY BE UNDEFINED ON SOME INPUTS.

TOTALLY COMPUTABLE

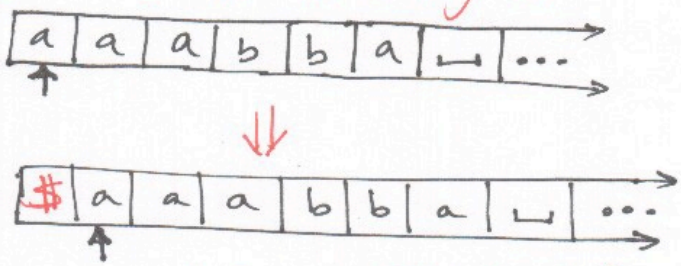
≡ DECIDABLE

DEFINED ON ALL INPUTS.

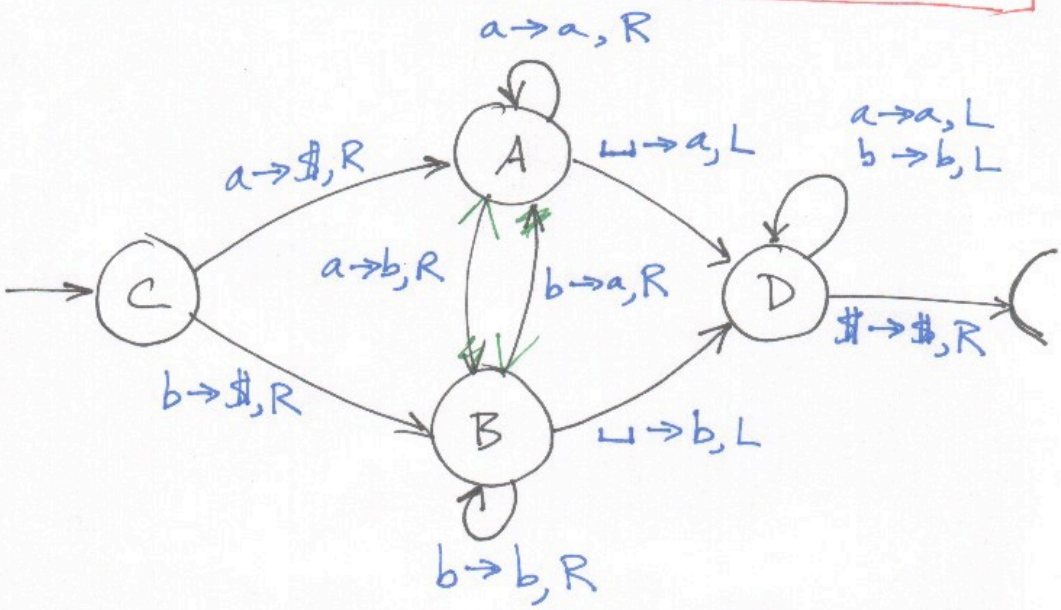
PROBLEM

How can we recognize the left end of the tape?

GOAL: 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 = (z+k)*n;
⇓
Algorithms
If $S \cap T = \emptyset$...
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 $S \cap T = \emptyset$...

EXAMPLE

Build a TM to ~~recognize~~ decide the language $0^N1^N0^N$.

ON THE EXAM YOU SHOWED THIS WAS NOT CONTEXT FREE, SO THIS WILL SHOW THAT THE "ONION" IS PROPER SUBSET.

IDEA We already have a TM to turn 0^N1^N into X^NY^N and to decide that language.

Use it as a "subroutine"!

STEP 1

000000 111 11 000000

↓

XXXXXX yyyyyy 000000

Reject if problems.

STEP 2 Build a similar TM to recognize y^N0^N

STEP 3 Build the final TM by "GLUING" these TMs together. into one large TM.

PROBLEM:

Compare two strings $\{ w\#w \mid w \in \{a,b,c\}^* \}$

Solution:

Use a new symbol, such as "X".

Turn the symbols into a "X" after they have been examined.

a a b a c # a a b a c

⇓

X a b a c # X a b a c

PROBLEM:

Do it non-destructively, without losing the 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 → x
b → y
c → z

a a b c a # a a b c a

⇓

x x y z a # x x y z a

$\Gamma = \{ a, \dot{a}, b, \dot{b}, c, \dot{c} \}$

Later, we can restore the strings if we need to.

a b c b # a b c b

⇓

$\dot{a}$ $\dot{b}$ $\dot{c}$ b # $\dot{a}$ $\dot{b}$ $\dot{c}$ b

⇓

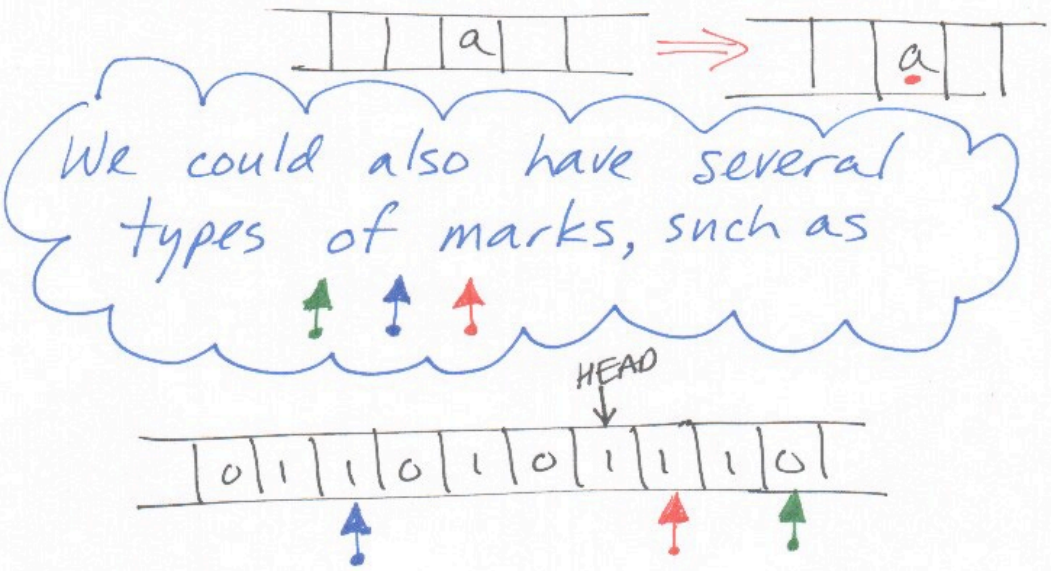
a b c b # a b c b

MARKING

NOW WE CAN SAY

"MARK THIS SYMBOL"

OR "PUT A DOT UNDER THIS SYMBOL."



THEOREM

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

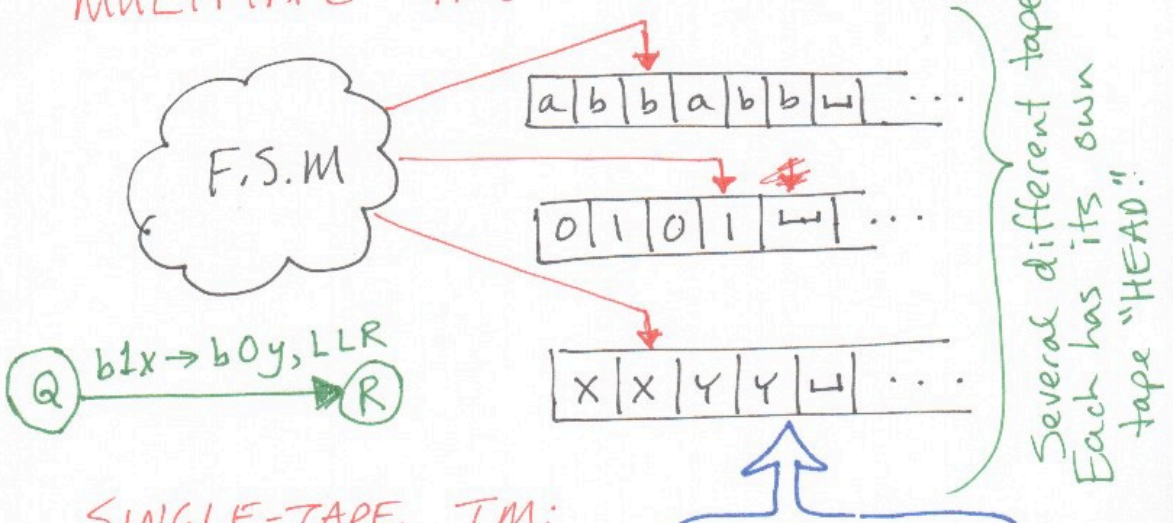
"EQUIVALENT" MEANS IT DECIDES/RECOGNIZES THE SAME LANGUAGES. IT'S NOT ABOUT SPEED, EFFICIENCY OR EASE OF PROGRAMMING.

PROOF

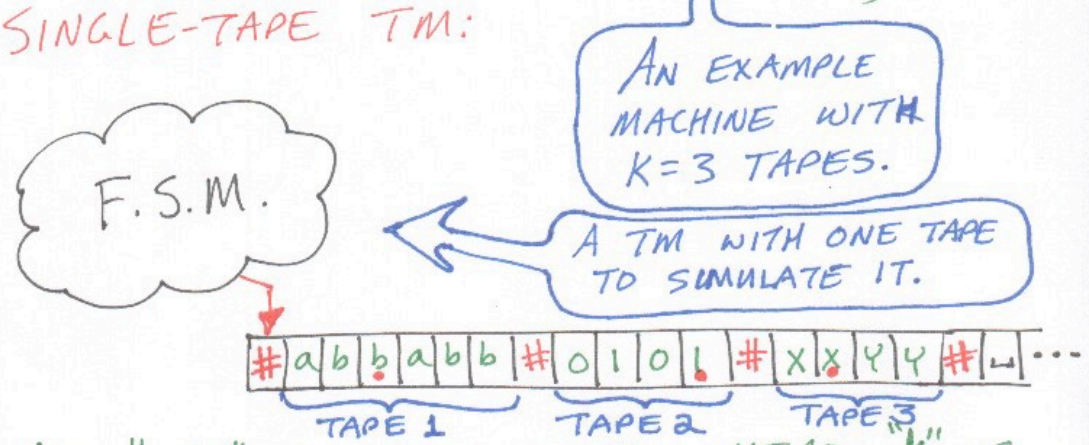
GIVEN A MULTITAPE TM, SHOW HOW TO BUILD A SINGLE-TAPE TM.

- 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

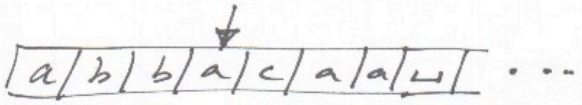
TRANSITION FUNCTION

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

A "CONFIGURATION" IS...

A STRING WHICH CAPTURES THE ENTIRE STATE OF A T.M. AT ONE MOMENT DURING A COMPUTATION

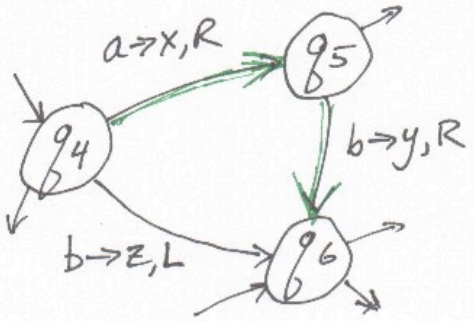
- STATE
- TAPE CONTENTS
- HEAD POSITION



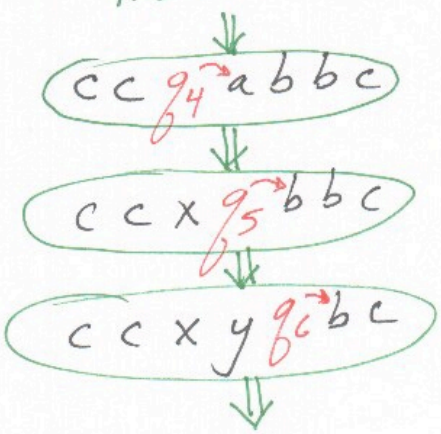
"abb q_{37} acaa"

AT EACH MOMENT IN THE COMPUTATION THERE CAN BE MORE THAN ONE SUCCESSOR CONFIGURATIONS!

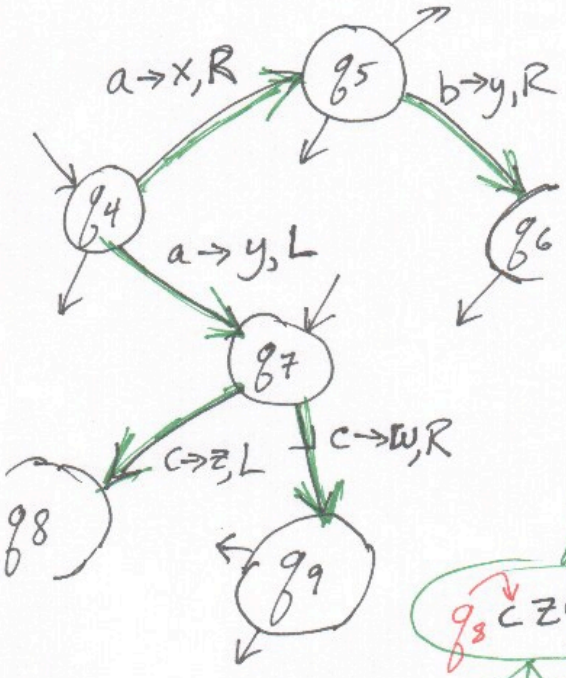
DETERMINISTIC TM

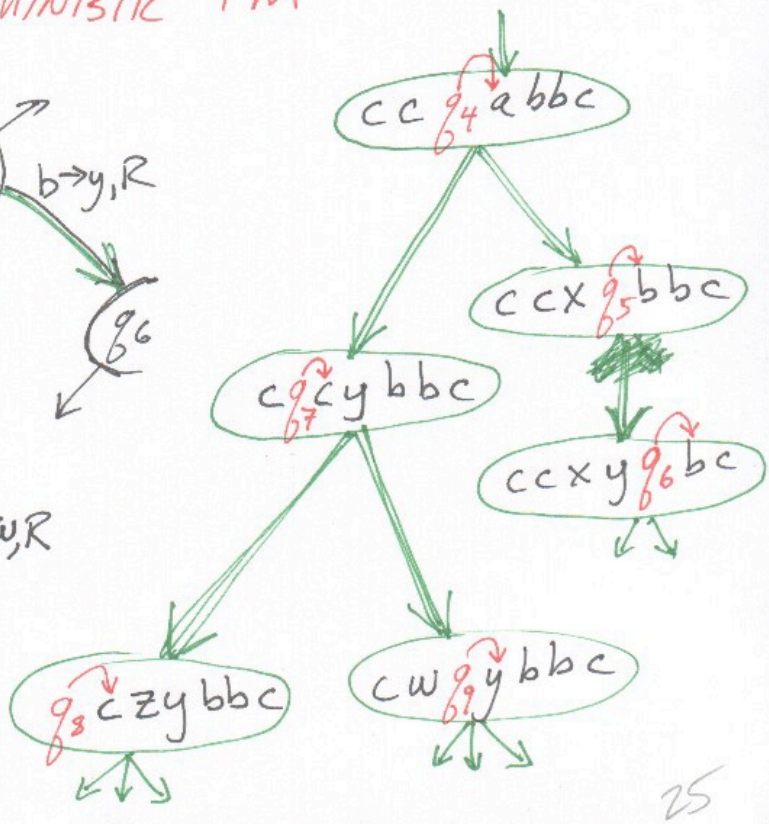


COMPUTATION HISTORY

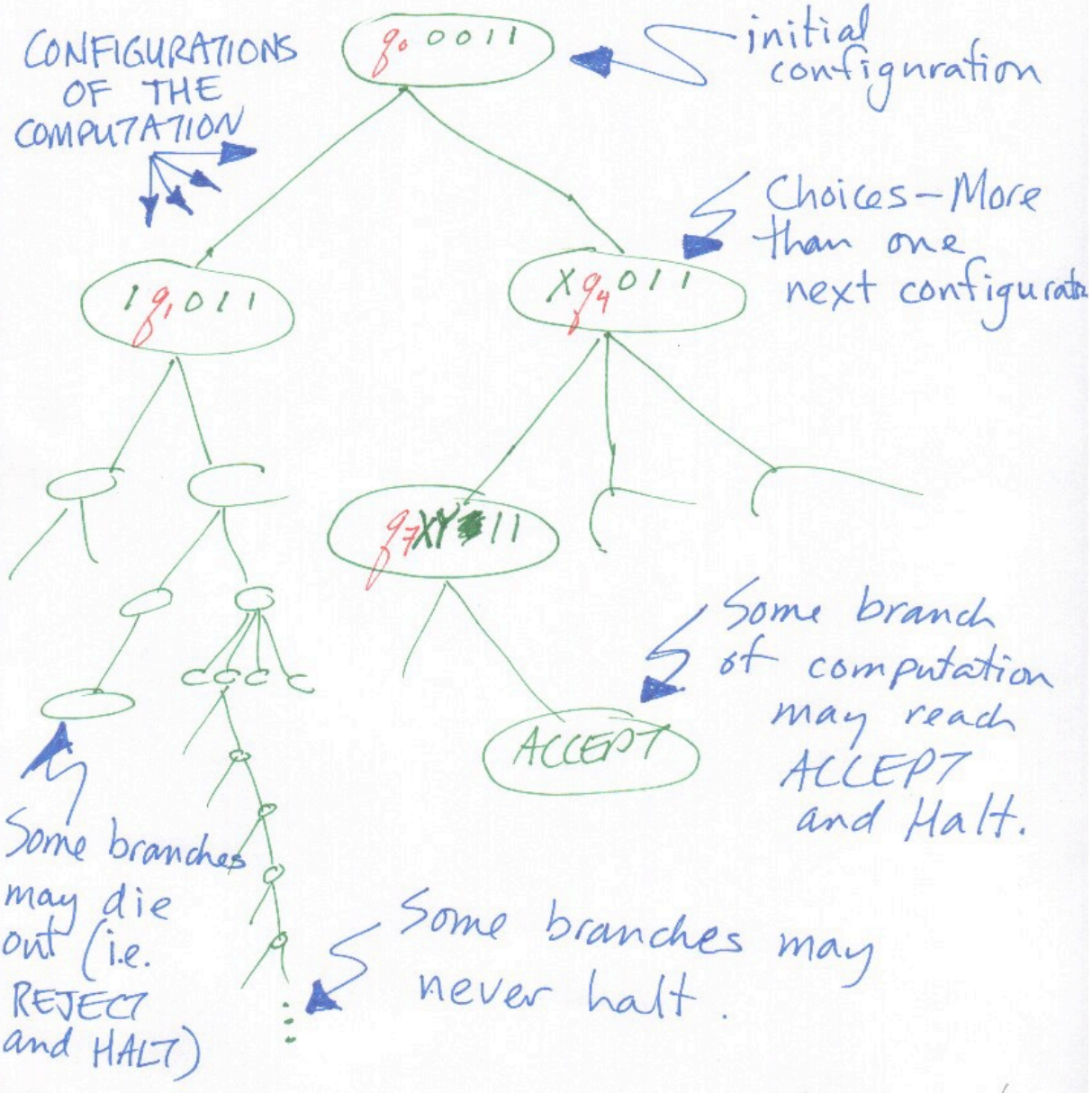


NONDETERMINISTIC TM





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.

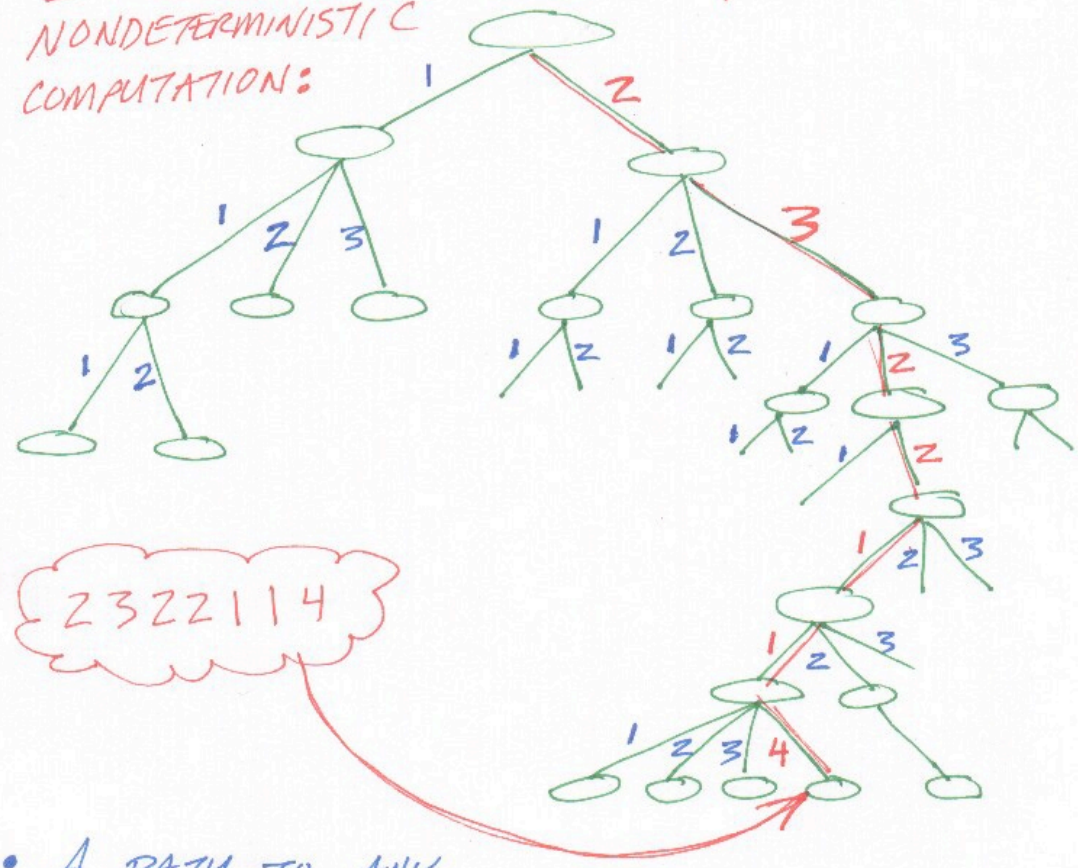
THEOREM

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 ALL BRANCHES OF COMPUTATION; SEARCH FOR ANY WAY N CAN ACCEPT.

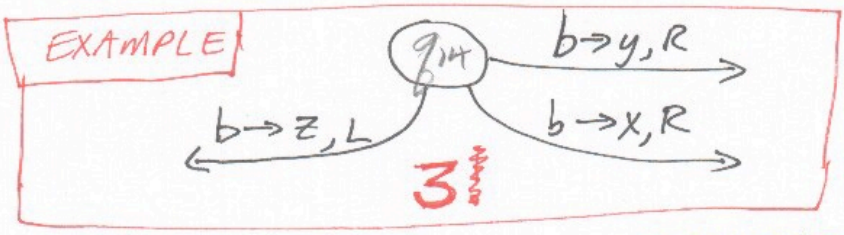
THE "COMPUTATION HISTORY" IS A TREE SHOWING ALL POSSIBLE BRANCHES/CHOICES IN A NONDETERMINISTIC COMPUTATION:

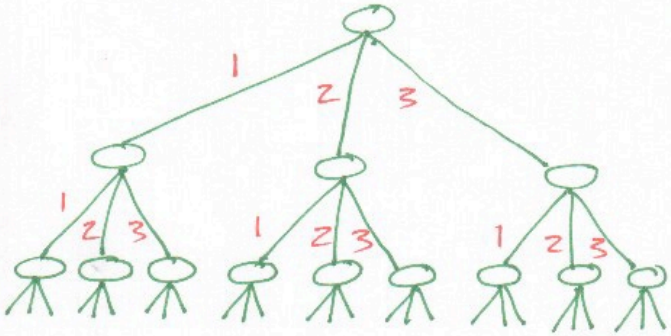


- A PATH TO ANY NODE IS GIVEN BY A NUMBER.
- SEARCH THE TREE, LOOKING FOR "ACCEPT."
- SEARCH ORDER?
  - DEPTH-FIRST? ← No!
  - BREADTH-FIRST! ← YES!
- TO EXAMINE A NODE:
  - PERFORM THE ENTIRE COMPUTATION FROM SCRATCH.
  - THE PATH NUMBERS TELL WHICH OF THE MANY NONDETERMINISTIC CHOICES TO MAKE.

HOW MANY CHOICES AT EACH STEP IN THE COMPUTATION?

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



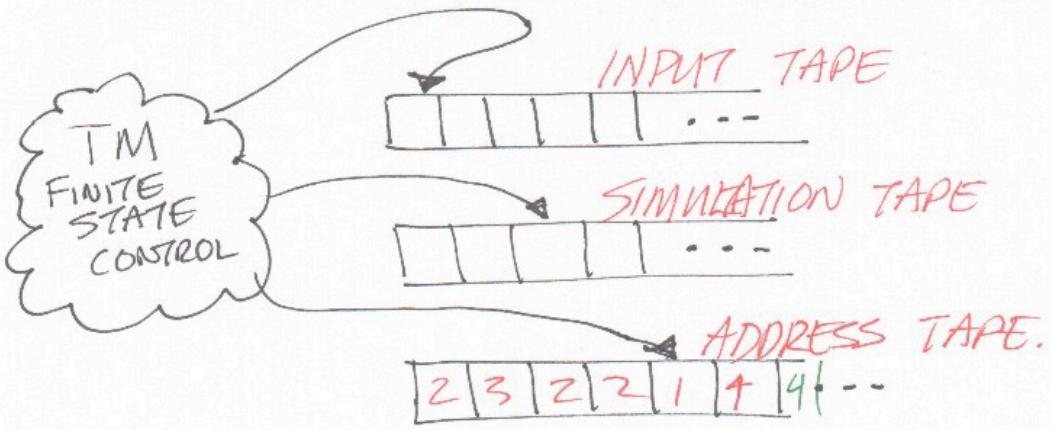


BREADT-FIRST SEARCH ORDER:

ε
1
2
3
11
12
13
21
22
23
31
32
33
111
112
113
121
122
123
⋮

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: LIKE THE TAPE OF A DETERMINISTIC T.M.

ADDRESS TAPE: USED TO CONTROL THE BREADTH-FIRST SEARCH.

- TELLS WHICH CHOICES TO MAKE DURING A SIMULATION.

ALGORITHM

INITIALLY: TAPE 1 CONTAINS THE INPUT.
TAPES 2 & 3 ARE EMPTY.

COPY TAPE 1 TO TAPE 2.

RUN THE SIMULATION.

- USE TAPE 2 AS "THE TAPE."
- WHEN CHOICES OCCUR, 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 GOES.

TRY THE NEXT CHOICE.
INCREMENT TAPE 3
REPEAT

or the computation dies out.

2 3 2 2 1 1 1
2 3 2 2 1 1 2
2 3 2 2 1 1 3
2 3 2 2 1 1 4
2 3 2 2 1 2 1

IF ACCEPT IS EVER ENCOUNTERED,
HALT AND ACCEPT.

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

HALTING?

IF A NON-DETERMINISTIC T.M. HALTS
ON ALL BRANCHES WITHOUT ACCEPTING,
THEN ~~IT~~ IT "REJECTS".

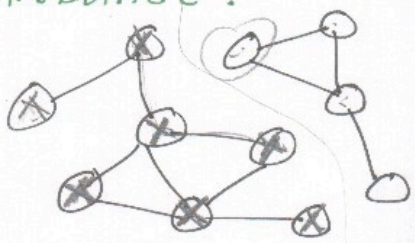
A LANGUAGE IS ~~TURING~~ "DECIDABLE"
IFF SOME NON-DETERMINISTIC
TURING MACHINE DECIDES IT.

"DECIDES":
Always HALTS! Will always
ACCEPT OR REJECT! Will
never LOOP!

EXAMPLE

CAN AN ARBITRARY PROBLEM BE EXPRESSED AS A LANGUAGE?

UNDIRECTED GRAPH



IS THE GRAPH "CONNECTED"?

WE MUST ENCODE THE PROBLEM. INTO A LANGUAGE:

$$A = \{ \langle G \rangle \mid G \text{ IS A CONNECTED GRAPH} \}$$

WE WANT:

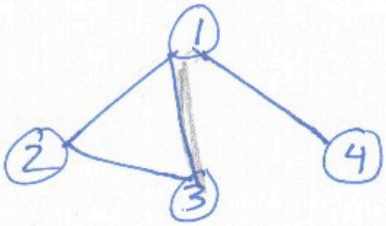
A TM TO DECIDE THIS LANGUAGE.

ACCEPT = "Yes, this is a connected graph"

REJECT = "No, Either it is not a valid representation of a graph OR the graph is UNCONNECTED."

LOOP? THIS PROBLEM IS DECIDABLE. THE T.M. WILL ALWAYS HALT.

ONE REASONABLE REPRESENTATION:



$\langle G \rangle =$

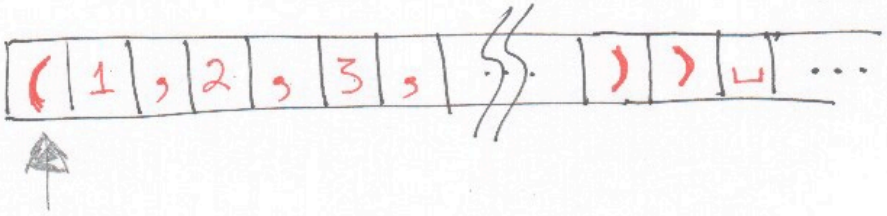
$(1,2,3,4)\big((1,2),(2,3),(1,3),(1,4)\big)$

List of node "names"

An edge from (1) to (3)

LIST OF EDGES

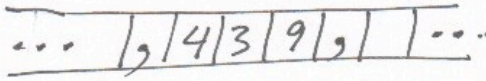
$\Sigma = \{(,), ,, 1, 2, 3, 4, \cdots, 9\}$



REPRESENTING NUMBERS ON A TAPE

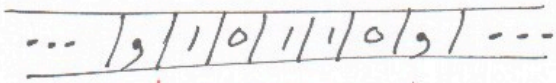
DECIMAL

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



BINARY

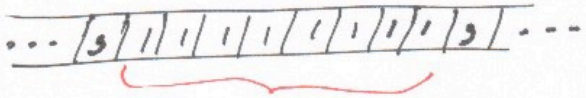
$\Sigma = \{0, 1, \cdots\}$



$= 22$

UNARY

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



$= 8$

(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 "C", 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 TM "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 ALL POSSIBLE STRINGS. in Σ^*.
IF M EVER ACCEPTS A STRING, THEN PRINT IT OUT.

PROBLEM?
M MIGHT LOOP ON SOME PARTICULAR STRING.

WE MUST RUN ALL THESE SIMULATIONS IN PARALLEL!

GOAL

RUN A TM ON ALL STRINGS IN Σ^*

SIMULTANEOUSLY / IN PARALLEL.

APPROACH

WE CAN LIST OUT ALL STRINGS IN $\Sigma^* = \{s_1, s_2, s_3, s_4, \cdots\}$

EG: ϵ, 0, 1, 00, 01, 10, 11, 000, 001, ...

THE COMPUTATION ON ANY STRING s_i MAY BE INFINITE.

WE MUST SOMEHOW INTERLEAVE THE DIFFERENT COMPUTATIONS.

WORK ON s_1 A LITTLE,

THEN WORK ON s_2 A LITTLE,

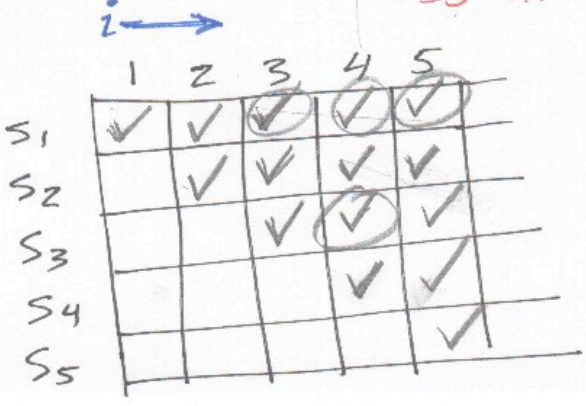
...

EVENTUALLY, MUST GO BACK AND DO MORE WORK ON s_1

ALGORITHM

```
FOR i = 1, 2, 3, ...   (INFINITE LOOP)
    FOR j = 1 to i
        SIMULATE TURING MACHINE M
            USE Sj AS INPUT.
            RUN SIMULATION FOR i STEPS
        IF M ACCEPTS Sj
            (... WITHIN i STEPS...)
        THEN PRINT Sj
    END
END
```

EXAMPLE: ASSUME M ACCEPT
S_1 AFTER **3** STEPS
S_3 AFTER 4 STEPS



OUTPUT:

S_1
S_1
S_3
S_1