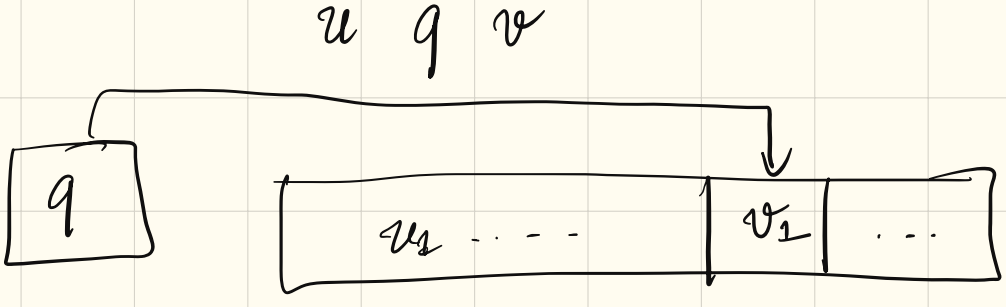
26 Mar 2023 – Theory of Computation

Turing Machines - Contd

Configuration = (current state, current contents of tape, head location)

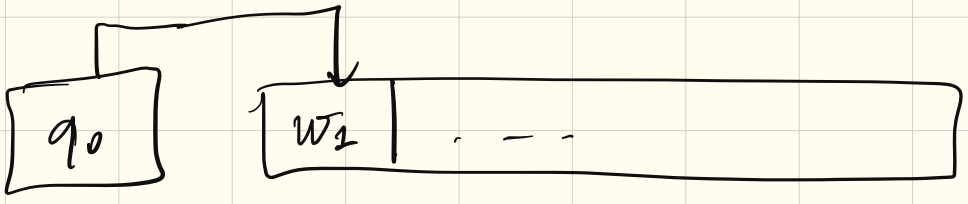
Config representation :



C_1 yields C_2 if TM can go from C_1 to C_2 in single step.

→ formal def^n of yields using δ

Start configuration: $q_0 w$



Accepting configuration: state of config is q_{accept}

Rejecting config: state q_{reject}

accepting & rejecting } halting configurations and do not yield any further configurations

Since machine is defined to halt when in states q_{accept} and q_{reject} we could have defined δ as

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

$$Q' = Q \setminus \{ q_{accept}, q_{reject} \}$$

A TM accepts input w if $\exists$ a sequence of configs $C_1, C_2, \ldots, C_k$ s.t

① C_1 – start config of M on input w

② each C_i yields C_{i+1}

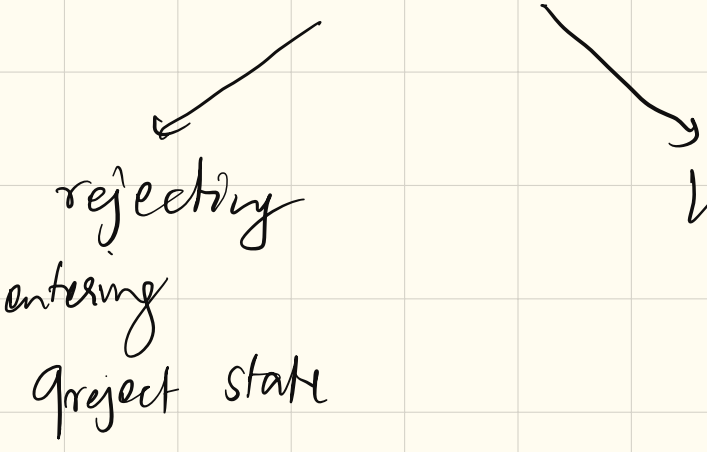
③ C_k is an accepting configuration.

Collection of strings M recognizes } Language of M } recognized by

$L(M)$

A language is called Turing-recognizable if some TM recognizes it.
aka recursively enumerable

M can Fail to accept an input



Sometimes, distinguishing a machine that is looping from one that is merely taking long is difficult

∴ We prefer machines that always halt → Decider

A decider that recognizes a language is said to decide that language

↓

Turing decidable or recursive

Every decidable language is turing recognizable

- examples : see tables

3.2 Variants of Turing Machine

Alternative defⁿ of TMs :

→ multiple tapes
→ non-determinism, etc

} variants of TM

TM and its variants } → same power recognize - the same class of language.

robustness : invariance of powers to certain changes in the defn

TM exhibits high degree of robustness

Multitape Turing Machine

→ ordinary TM + several tapes

→ each tape has its own head for reading and writing.

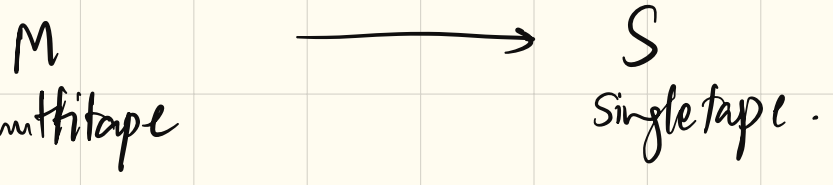
→ Initial input : tape 1
others blank.

$$\delta : Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{L, R, S\}^k$$

k = no. of tapes.

two machines are equivalent if they recognize the same language.

Every multitape TM has an equivalent single-tape TM.



Key idea : simulate M with S.

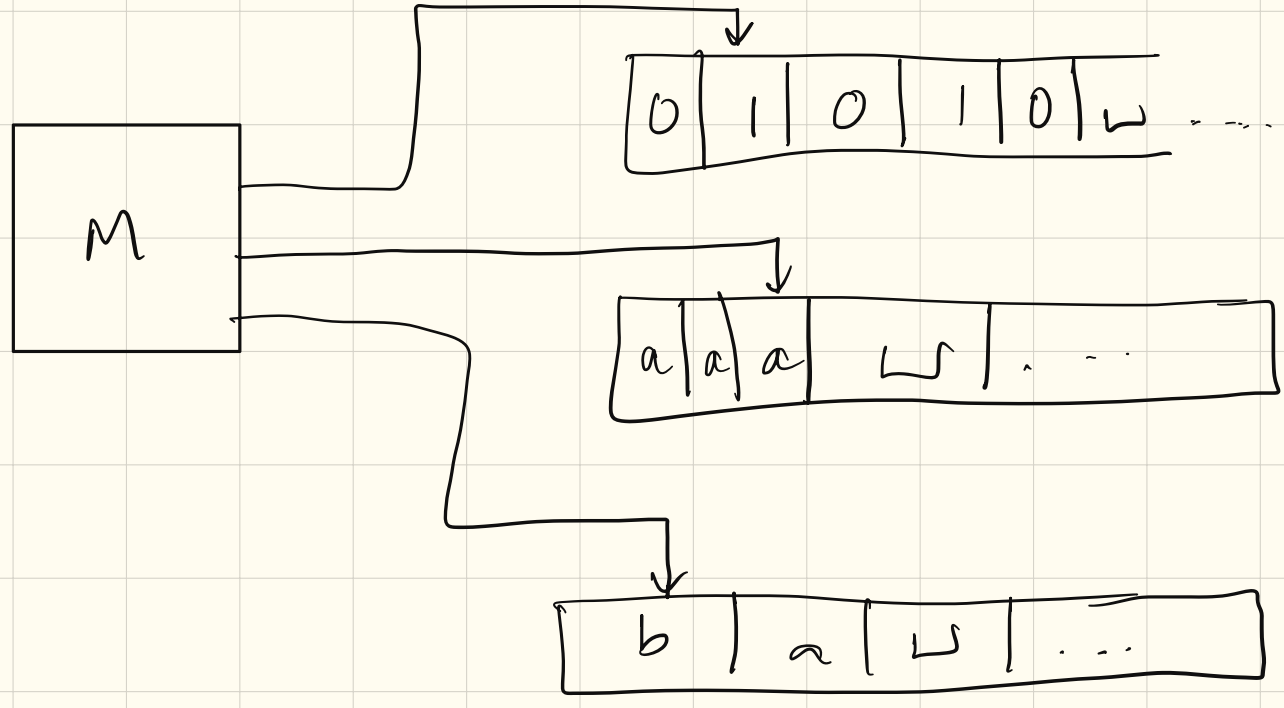
Let M have k tapes

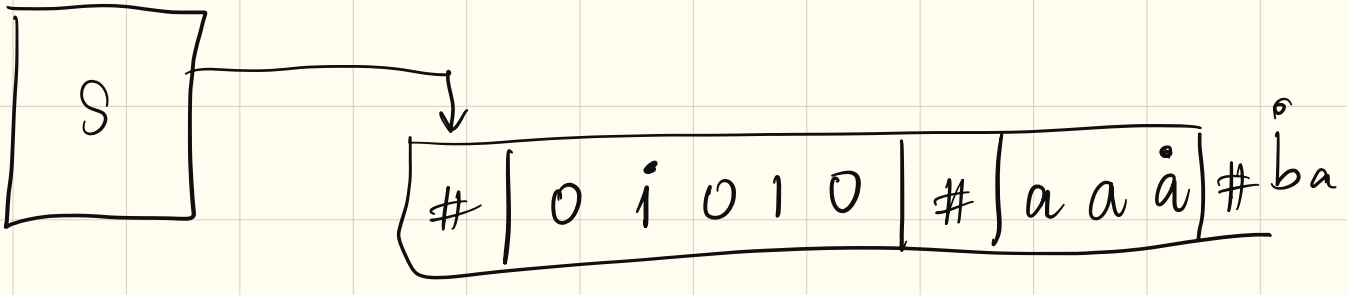
S simulates the effect of k tapes by storing their info on its single tape.

(#) → we as delimeters to separate the contents of diff. tapes.

Keep track of location of heads:

Write a tape symbol with a dot above it to mark the place where head would be





S = " On input $w_1 \cdots w_n$:

1. $\# \dot{w}_1 w_2 \cdots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \cdots \#$

2. To simulate a single move, S

scan from the first # (left end) to the $(k+1)^{th}$ # right end.

Then make a pass to update the tapes according to the way M's transition fn dictates

3. move # to right

$\Rightarrow$ M has moved the corresponding head onto the previously unread blank portion.

∴ A language is turing-recognizable if and only if some multitaple TM recognizes it.

⟶ Single tape is special case of Multitapé above discuss'n ⟵

Non-deterministic turing machines

At any point in a comptation, the machine may proceed according to several possibilities.

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

Computation of an NTM ⟶ tree whose branches correspond to different possibilities for the machine

Every nondeterministic TM has an equivalent deterministic TM

(N)

Proof Idea: Simulate non-deterministic TM with deterministic TM.

(D)

Have D try all possible branches of N's non-deterministic computation.

If D ever finds the accept state on one of these branches, D accepts

→ We view N's computation on input w as a tree.

→ Each branch represents one of the branches of non-determinism

→ Each node = config.
root = start config.

(D) → searches this tree for an accepting configuration

Bad idea ⇒ Depth-first search
↓
could go forever down one branch before backing up to explore other branches

∴ Design D to use BFS
ensures that D will visit every node in the tree until it encounters an accepting config.

Proof: The simulating TM D has 3 tapes, By Theorem 3.13, this arrangement is equivalent to having a single tap.

Tape 1: contains the input string and is never altered

Tape 2: maintains a copy of N's tape on some branch of its non-deterministic computation.

Tape 3: keeps track of D's location in N's non-deterministic computation tree.

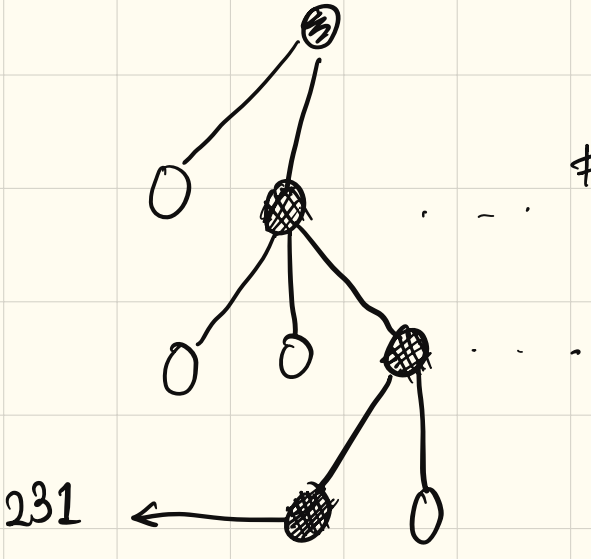
Tape 3: ⟶ every node in the tree can have at most b children

↳ b = size of the largest set of possible choices given by N's δ

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

To every node in the tree we assign an address over

$$\Gamma_b = \{1, 2, \ldots, b\}$$



some choices may not be available

Tape 3 contains a string over Γ_b

Empty string → address of root

D:

1. Initially Tape 1 contains the input w, tapes 2 and 3 are empty.

2. Copy tape 1 to tape 2 and initialize the string on tape 3 to be ε.

3. Use tape 2 to simulate N with input w on one branch of its non-deterministic computation.
   Before each step of N, consult the next symbol

on tape 3: to determine which choice to make among those allowed by N's δ. If no more more symbols remain on tape 3 or if this Non-det choice is invalid,

or reject ← abort this branch by going to stage 4

If accepting configuration is encountered, accept the input.

4. Replace string on tape 3 by next ordered string. Simulate the next branch of N's computation

A language is turing-recognizable if and only if some non-det. TM recognizes it.

$\longrightarrow$

Any det. TM is non-det TM

$\longleftarrow$

Thm 3.16

N always halts $\Longrightarrow$ D always halts

A non-deterministic turing machine is a <u>decider</u> if all branches halt on all inputs.

A language is decidable if and only if some non-det TM decides it.

Exercises

3.1

Here we describe a Turing machine (TM) M_2 that decides $A = \{0^{2^n} \mid n \geq 0\}$, the language consisting of all strings of 0s whose length is a power of 2.

M_2 = "On input string w:

1. Sweep left to right across the tape, crossing off every other 0.
2. If in stage 1 the tape contained a single 0, *accept*.
3. If in stage 1 the tape contained more than a single 0 and the number of 0s was odd, *reject*.
4. Return the head to the left-hand end of the tape.
5. Go to stage 1."

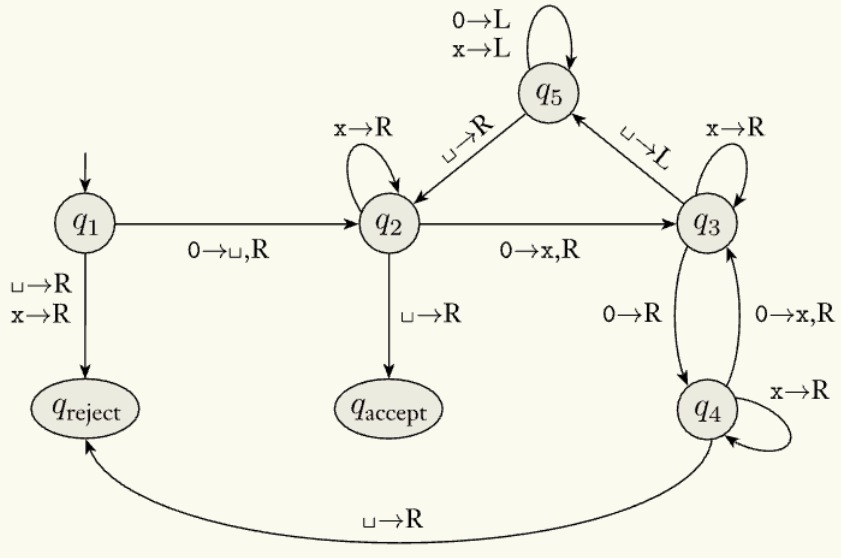


FIGURE 3.8
State diagram for Turing machine M_2

This exercise concerns TM M_2, whose description and state diagram appear in Example 3.7. In each of the parts, give the sequence of configurations that M_2 enters when started on the indicated input string.

a. 0.

[A] **b.** 00.

c. 000.

d. 000000.

a. $q0 \longrightarrow 0\,q_2 \longrightarrow 0\sqcup q_{accept}$ (X)

$q0 \longrightarrow \sqcup\, q_2 \sqcup \longrightarrow$

$\sqcup\sqcup\, q_{accept}\sqcup$

b. 00

$q_100 \longrightarrow \sqcup\, q_2\, 0 \longrightarrow \sqcup x\, q_3\, \sqcup$

$\downarrow$

$q_5 \sqcup x \sqcup \longleftarrow \sqcup\, q_5\, x\, \sqcup$

$\downarrow$

$\sqcup\, q_2\, x \longrightarrow \sqcup\, x\, q_2$

$\downarrow$

$\sqcup x \sqcup\, q_{accept}$

c. 000

$q_1 000$

$\sqcup q_2 00$

$\sqcup x q_3 0$

$\sqcup x 0 q_4 \sqcup$ $\quad$ $0 \sqcup x 0 \sqcup q_{reject}$

d. 000000

$q_1 000000$

$\sqcup q_2 0000$

$\sqcup x q_3 000$

$\sqcup x 0 q_4 00$

$\sqcup x 0 x q_3 0$

$\sqcup x 0 x 0 q_4$

$\sqcup x 0 x 0 x q_3 \sqcup$

$\sqcup x 0 x 0 q_5 \sqcup$

$\sqcup x 0 x \quad q_5 0$

⋮

$\sqcup q_2 x 0 x 0 \sqcup$

$\sqcup x q_3 0 x 0$

$\sqcup x 0 q_4 x 0$

$\sqcup x 0 x q_4 0$

$\sqcup$ ⋱

EXAMPLE 3.9

The following is a formal description of $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, q_{\text{accept}}, q_{\text{reject}})$, the Turing machine that we informally described (page 167) for deciding the language $B = \{w\#w \mid w \in \{0,1\}^*\}$.

- $Q = \{q_1, \ldots, q_8, q_{\text{accept}}, q_{\text{reject}}\}$,
- $\Sigma = \{0,1,\#\}$, and $\Gamma = \{0,1,\#,\mathrm{x},\sqcup\}$.
- We describe δ with a state diagram (see the following figure).
- The start, accept, and reject states are q_1, q_{accept}, and q_{reject}, respectively.

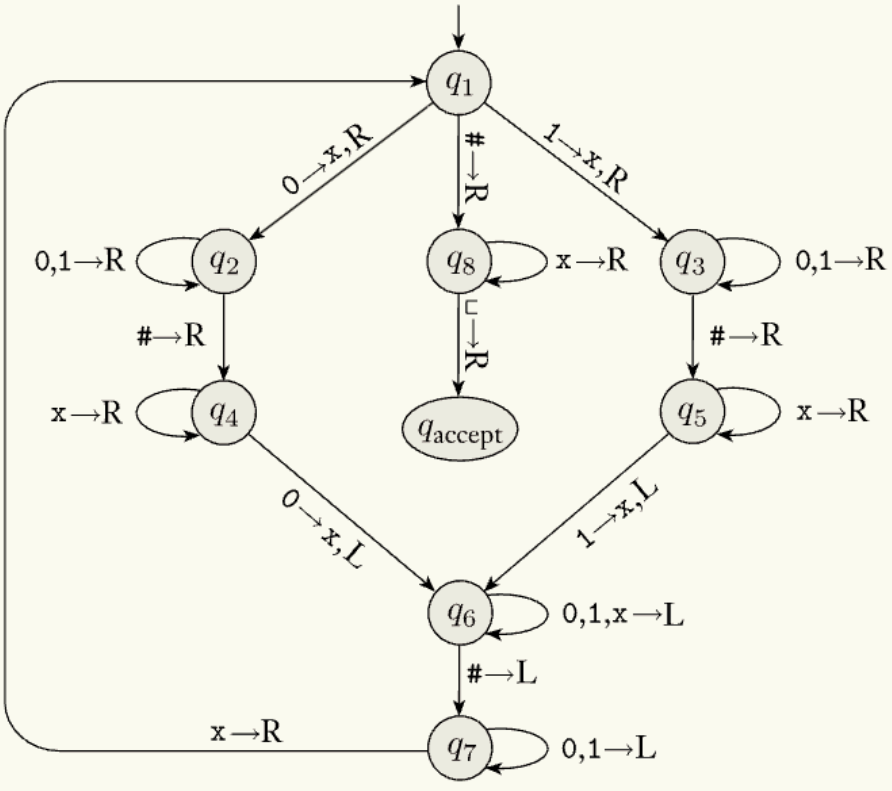


FIGURE 3.10
State diagram for Turing machine M_1

$q_1 10\#11$

$\mathrm{x}\, q_3 0\#11$

$\mathrm{x}\, 0\, q_3 \#11$

$\mathrm{x}\, 0\#\, q_5 11$

$\mathrm{x}\, 0\, q_6 \#\, \mathrm{x}\, 1$

$\mathrm{x}\, q_7\, 0\#\, \mathrm{x}\, 1$

$q_7\, \mathrm{x}\, 0\#\, \mathrm{x}\, 1$

$\mathrm{x}\, q_1\, 0\#\, \mathrm{x}\, 1$

$\mathrm{x}\,\mathrm{x}\, q_2 \#\, \mathrm{x}\, 1$

$\mathrm{x}\,\mathrm{x}\,\#\, q_4\, \mathrm{x}\, 1$

$\mathrm{x}\,\mathrm{x}\#\mathrm{x}\, q_5 1$

3.2 This exercise concerns TM M_1, whose description and state diagram appear in Example 3.9. In each of the parts, give the sequence of configurations that M_1 enters when started on the indicated input string.

[A]**a.** `11`.

b. `1#1`.

c. `1##1`.

d. `10#11`.

e. `10#10`.

a. 11

$q_1\ 11$ $\quad\quad$ $x\ 1\ \sqcup$ q_{reject}

$x\ q_3\ 1$

$x\ 1\ q_3\ \sqcup$

b. 1#1

$q_1\ 1\ \#\ 1$	$x\ q_6\ \#\ x$	$x\ \#\ q_1\ x$
$x\ q_3\ \#\ 1$	$q_7\ x\ \#\ x$	$x\ \#\ x\ q_8\ \sqcup$
$x\ \#\ q_5\ 1$	$x\ q_2\ \#\ x$	$x\ \#\ x\ \sqcup\ q_{accept}$

c. 1##1

Enumerators

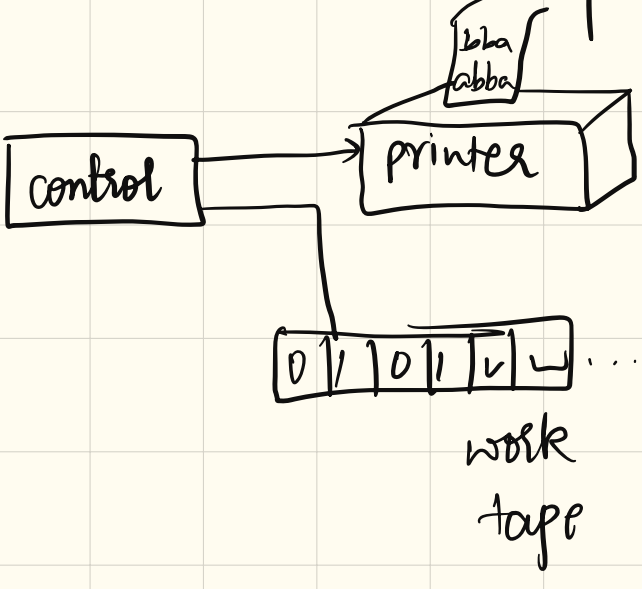
recursively-enumerable ≡ turing-recognizable

originates from a TM variant called

Enumerator

→ turing machine with an attached printer.

→ can use the printer as an output device to print strings.



List: each time the TM wants to add an output to the list, it sends the string to the printer

Enumerator → starts with a blank input on its work tape.

(E)

→ doesn't halt ⟹ ~~may~~ print an infinite list

Language enumerated by E → collection of all strings it eventually prints out.

→ any order

→ repitions possible

Thm

A language is Turing-Recognizable iff some enumerator enumerates it.

⟸ If we have an enumerator E that enumerates a lang. A, ∃ a TM M that recognizes A.

M: On input w:

1. Run E. Every time that E outputs a string, compare it with w.
2. If w ever appears in the output of E, accept.

Clearly, M accepts those strings that appear on E's list.

($\rightarrow$) If a TM M recognizes a language A, we can construct the following enumerator E for A.

Say that $s_1, s_2, \ldots$ is a list of all possible strings in Σ^*.

E = " Ignore the input

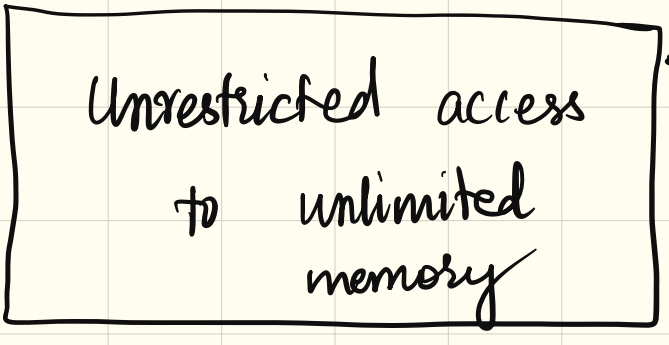
1. Repeat for $i = 1, 2, 3, \ldots$
    1. Run M for i steps on each input $s_1, s_2, \ldots, s_i$
    2. If any computations accept, print out the corresponding s_j. "

Equivalence with other models

→ Many other models of general purpose computation have been proposed.

Some are very different.

All share the essential feature of TMs



← distinguishes from weaker models like FSA and Pushdown Automata

Analogy - algorithm in one language and not other?

No

We can compile LISP into Pascal and Pascal into LISP.

⇓

the two languages describe exactly the same class of algorithms

So do all reasonable PLs.

The widespread equivalence of computational models holds for the same reason → one can simulate the other

equivalence → important philosophical corollary

We can imagine many different computational models, but the class of algorithms they describe remains the same.

Underlying class of algorithms

↓

same

The definition of algorithm

→ informal defn

→ procedures or recipes.

→ important role in mathematics.

finding prime

GCD, etc.

Precise notion of algorithm was not developed until 20th century.

Why do we need a precise definition?

Hilbert's Problems

ICM Paris

→ 23 math problems as challenge.

10th problem. → algorithms.

devise an algorithm that tests whether a polynomial has an integral root.

described as "process according to which it can be determined in finite no. of operations"

assumed that it must exist → someone need only find it.

No algorithm exists for this task

↓

algorithmically unsolvable. } Would have been impossible to come up with this conclusion with their intuitive concept of algorithm.

intuitive concept → adequate for giving algorithms to certain tasks

↓ useless for showing that no algorithm exists for a particular task

Proving that an algorithm does not exist requires having clear defⁿ of algorithm.

Defⁿ → 1936 papers of
Alonzo Church and
Alan Turing

Church used a notational system
called λ- calculus
to define algorithm.

Turing → machines. → equivalent

Church-Turing Thesis → connection b/w informal notion of algorithm and precise defⁿ

provides the defⁿ of algorithm
to resolve 10th problem

1970 → showed that no algorithm exists for testing whether a polynomial has Z roots.

people

$D = \{ p \mid p$ is a polynomial (multivariable or single) with an integral root $\}$

Is D decidable?

No

D is turing-recognizable.

$D_1 = \{ p \mid p$ is a polynomial over x with an integral root $\}$

TM M_1 that recognizes D_2:

M_2 = "On input $\langle p \rangle$: where p is a polynomial over the var x.

1. Evaluate p with $x = 0, 1, -1, 2, \ldots$
   If at any point, p evaluates to zero, accept.

$M_1 \rightarrow$ may run forever.

Similarly M

M_2 and $M \longrightarrow$ recognizers, but not deciders.

$M_1 \longrightarrow$ can be converted to a decider because we can get bounds for the roots $\rightarrow$ restricts the search.

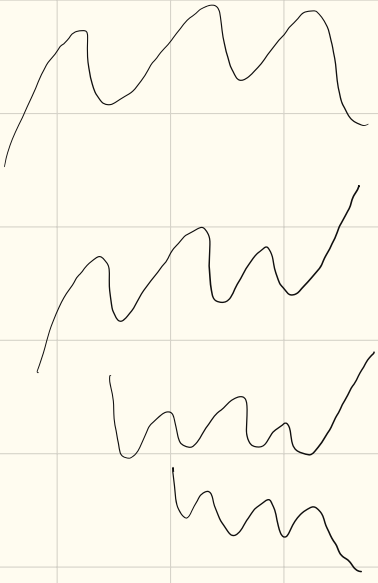
Roots of a single-variable polynomial must lie b/w

$$\pm \; k \frac{C_{max}}{C_1}$$

$k \rightarrow$ no. of terms in the polynomial.

$C_{max} \rightarrow$ coefficient with the largest absolute value.

a_1 → coefficient of the highest order term



If a root is not found within these bounds, the machine rejects.

Matijasevic's theorem

↳ calculating such bounds for multivariable polynomials is impossible.

Terminology for Describing TMs

→ TMs merely serve as a precise model for the definition of algorithm.

→ What is the right level of detail?

① Formal description
- → TM states
- → δ, etc.
- → lowest, most detailed

② Implementation description
- → english prose
- → describe how TM moves its head and the way it stores data

③ High-level description,
- → english prose to describe an algorithm.

Format and notation

Input to the TM ⟶ always a string.

Any other object ⟶ represent as a string

Encoding of an object into its rep representation as a string : $O \longrightarrow \langle O \rangle$

Several objects: $O_1, O_2, \ldots, O_k$

↓

single encoding

$\langle O_1, O_2, \ldots, O_k \rangle$

any encoding works ☺

Algorithm description:

1) indented

2) 1st line: describes input

w ⇝ taken to be a string

$\langle A \rangle$ ⇝ implicitly test if encoding of the object is in desired form / reject if not.

Example

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

high-level description of a TM M that decides A.

M = " On input $\langle G \rangle$, the encoding of graph G:

1. Select the first node of G and mark it.
2. Repeat the following stage until no new nodes are marked:
3. For each node in G, mark it if it is attached by an edge to a node that is already marked

4. Scan all the nodes of G to determine whether they are all marked.
If they are → accept
o/w reject.

Encoding of G ✓
Checking ✓

Is it worse than actually trying and solving?
YES
Do I have time? NO

EXERCISES
↓

Note: I did not get time to actually solve these because of the fast paced nature of
- study
- eat
- entertainment
- work

Used quizlet to just understand what's happening

fuck, almost everything.
And I can do nothing about it (at least for ~~this~~ exams)

Exercises

3.3

Thm 3.16 $\longrightarrow$ Det TM $\equiv$ NTM

Corollary 3.19

3.4 Formal defn of enumerator

7-tuple: $(Q, \Sigma, \Gamma, \delta, q_0, q_{print}, q_{halt})$

- Q: finite set of states
- Σ: output alphabet
- Γ: tape

$$\delta : Q \times \Gamma^2 \longrightarrow Q \times \Gamma^2 \times \{L, R\}^2$$

$q_0 \longrightarrow$ initial

$q_{print} \longrightarrow$ print $\rightsquigarrow$ if content of second tape is $w \sqcup \ldots$, $w \in \Sigma^*$ } we say it is printed

$q_{halt} \longrightarrow$ halting

3.5 a) Yes?

b) Yes ✗ ⊔ ∈ T always

c) No ✗ uhh → left end

d) No qaccept and qreject

3.6 Because it might never halt for some s_k

3.7

3.7 Explain why the following is not a description of a legitimate Turing machine.

M_{bad} = "On input $\langle p \rangle$, a polynomial over variables $x_1, \ldots, x_k$:

1. Try all possible settings of $x_1, \ldots, x_k$ to integer values.
2. Evaluate p on all of these settings.
3. If any of these settings evaluates to 0, *accept*; otherwise, *reject*."

~~It is still a legitimate TM~~

Step 2 can never reach step 3

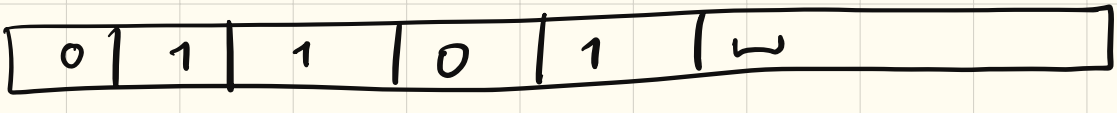
↳ so what ?

1

Looping on its own is obviously not a problem, however this machine puts its accept and reject states after a never ending search through infinite combination of integers, so it won't halt on any input at all.

3.8

(a)



find the first ␣ → write E

while (there is a 0 or 1

On input string w:

1. Scan the tape and mark the first 0 that has not been marked.
   If no unmarked 0 is found, go to stage 4.
   O/w move head back to the front

2. Scan the tape and mark the first 1 that has not been marked. If no unmarked 1 is found, reject.

3. Move the head back to the front of the tape and go

to stage 1.

4. Move the head back to the front of the tape. Scan the tape to see if any unmarked 1s remain. If none are found, accept; otherwise, reject

b. find leftmost 1 and then try to find two 0s

c. Swap accept and reject.

3.9

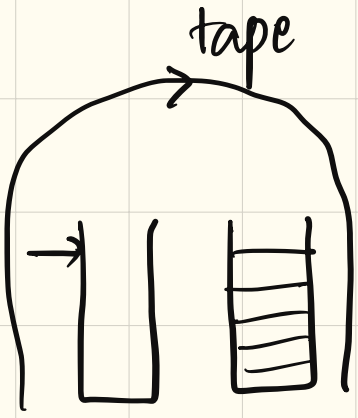
a. 2-PDAs are more powerful than 1-PDAs

Simulate tape of Turing machine using two stacks

top of 1st stack = position of head of TM

$$\delta(q, \alpha) = (r, \beta, R)$$

is simulated by popping α and pushing β and moving symbol from top of second stack to the top of first (if empty, push $\sqcup$)



$\delta(q, \beta) = (r, \alpha, L)$

similar

pop from 1st stack, place into second

This proves that 2-PDAs are atleast as powerful as TMs

can recognize

$a^n b^n c^n$

1-PDA

cannot

Part b

TM can easily simulate 2-PDA,

3-PDA, k-PDA

↳ using k-tapes

each tape ⇒ copy of string on i^{th} stack

∴ All k-PDAs $k > 1$ are equivalent

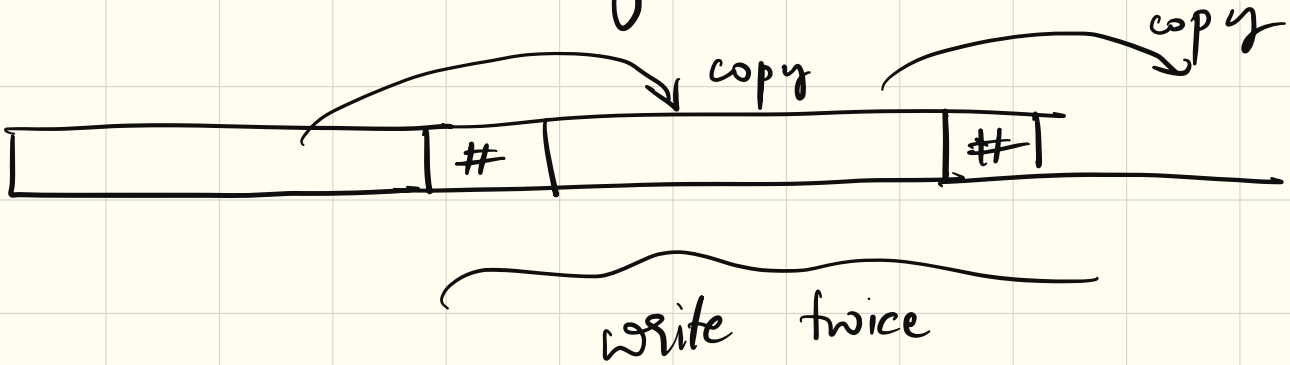
3.10

Idea: Try to simulate ordinary TM with this restricted TM.

instead of altering a tape directly,

transfer the whole string to another portion of tape, still unfonched,

and write the altered string there



↓

maintain 2 cells for each cell,

one for content, one for copied or not.

3.11