Turing Machines.

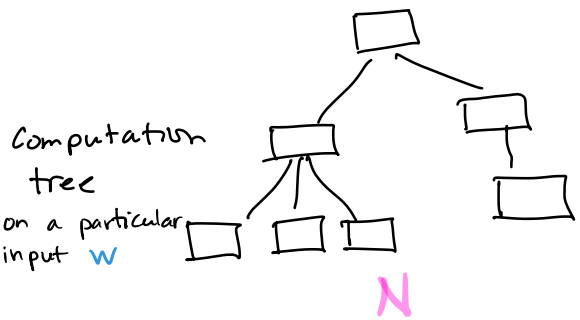
Does non-determinism add computational power to the TM model?
I.e. can you always come up with a deterministic equivalent?

Eg:

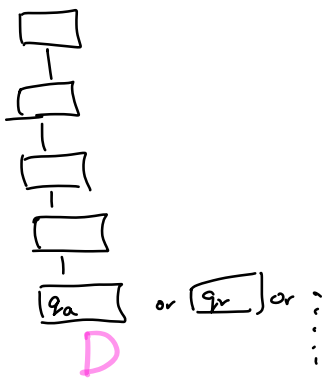
Is it composite?
- "Lucky Guess" two numbers x, y (on a separate tape)
- multiply $x \times y$
- check that the result is equal to the input.

Theorem:

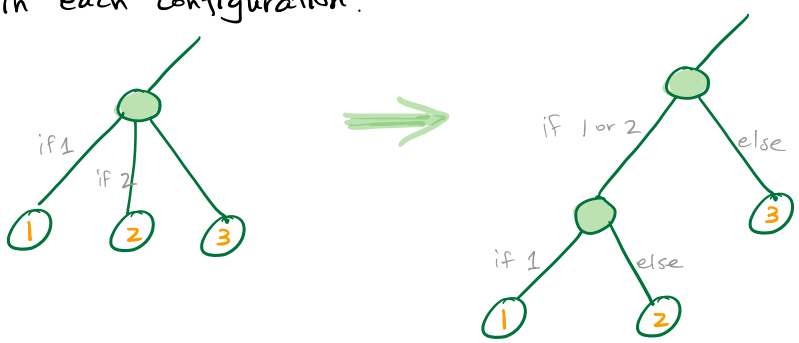
Proof: recall: non-det TM



det-TM .



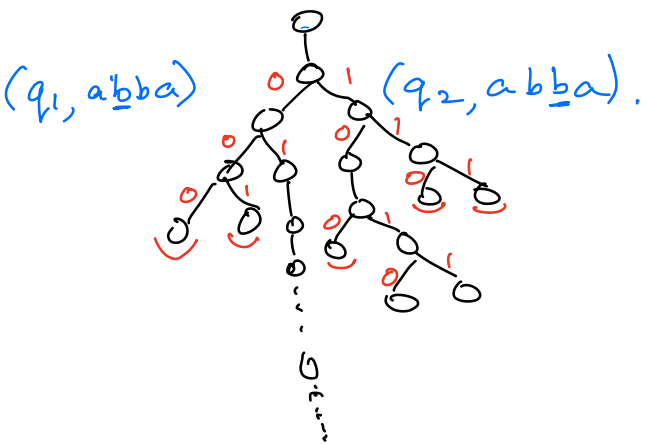
We can convert each non-det TM computation so it has ≤ 2 choices (branches) of what to do in each configuration.



D is det: On input w

- produce N's computation tree, as N would operate on w.

Consider N's computation tree on input w



First attempt at a D that simulates M:

- Just go L
  - if reaches h_r - REJECT
  - if reaches h_a - ACCEPT

  why doesn't this work?

Attempt #2

- Just go L
- if reaches a h_r, back up to lowest choice to go L and go R instead

why doesn't this work?

3rd Attempt:

- D will enumerate strings over {0,1} in shortlex order

  ε, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, ...
- For each such string, D will use the bits as "cointosses" deciding whether to go L (0) or R (1) as it simulates traversing a computational path in N's computation tree.

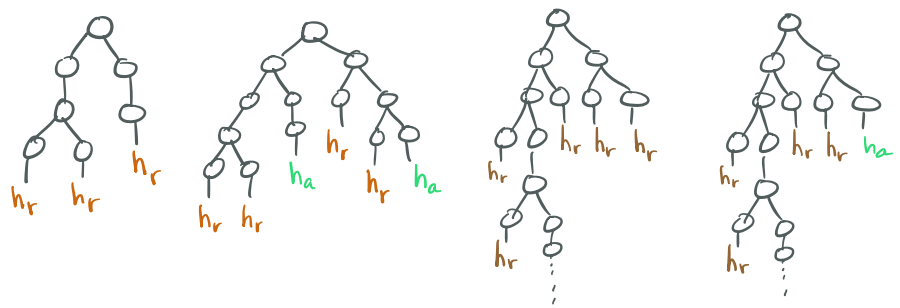
I.e., D uses BFS to explore N's computation tree, looking for _____

If it exists in the tree at all, it exists a finite distance down the tree, and so D will eventually find it and will halt and accept.

If _____ is not in the tree and the tree is infinite,

D will not halt on input w

But then .... what was N's behaviour on w?

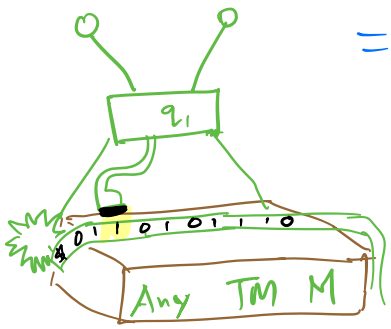


Mar 10, '26
Mar 16 '23
Mar 13 25

A Universal TM $\mathcal{U}$

One TM $\mathcal{U}$ can take a TM description $\langle M \rangle$ (encoding of M) and an input string $\langle w \rangle$ on its input tape, and simulate the computation of M on w. (!!!)

M on w.



$$= \langle \{q_0, q_1, q_2, h_a, h_r\}, \{a,b\}, \{a,b,\sqcup,X\}, \{(q_0,a,q_2,b,R), (q_0,b,q_1,b,R), \dots\}, q_0, h_a, h_r \rangle$$

(Q, Σ, Γ, δ)

$\mathcal{U}$ = "on input $\langle M, w \rangle$:

tape 1. [a b a b, ⊔ ⊔] ← treated as M's tape (w)

tape 2. [q_3 #] ← tells M's current state.

tape 3. [$(q_1, a, q_2, b, L), \dots$] ← δ_M M's transition function.

1. if tape 2 contains q_r, REJECT.
   (q_r: M's reject state.)

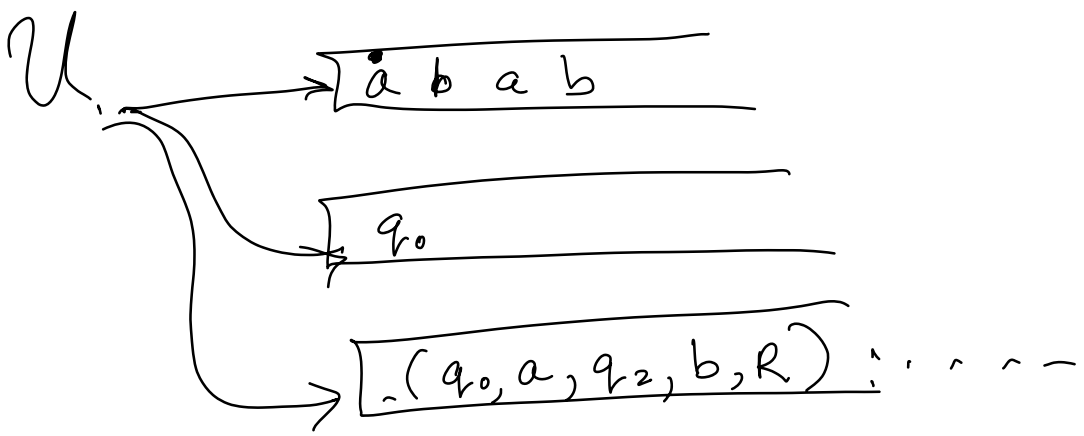
if tape 2 contains $q_1 a$, ACCEPT.

2. Otherwise, scan L on Tape 3 until it finds the right transition for M to make, and update Tape 1+2 accordingly, and go to step 1."

$U(\langle M, w \rangle) = M(\langle w \rangle)$

"the result of running U on input $\langle M, w \rangle$."

(Notation: $M(\langle w \rangle)$ = ACCEPT - if M accepts w
REJECT - if M rejects w
loop - if M loops on w.



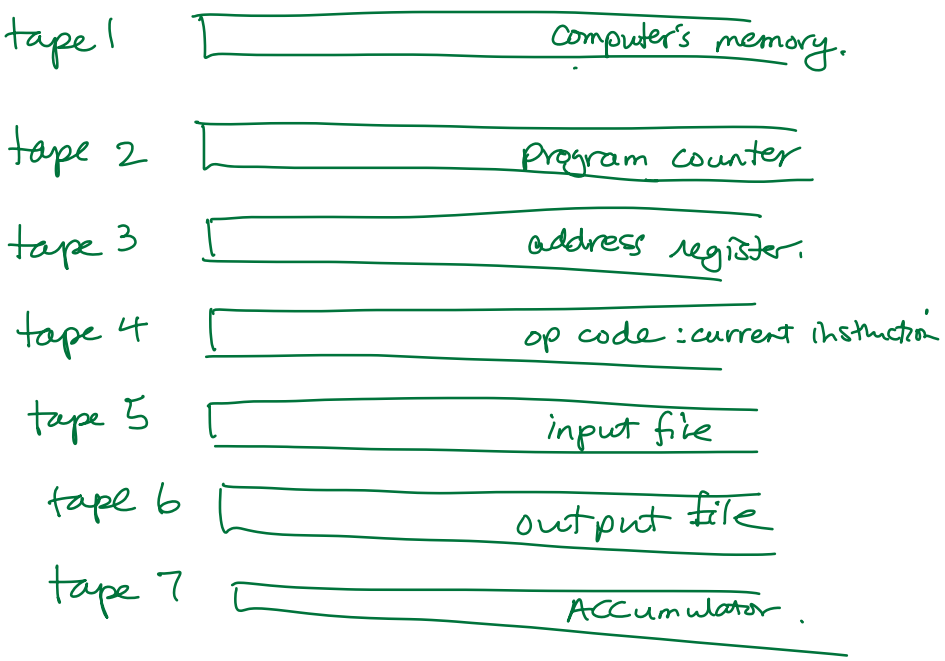
Simulation of a Real Computer.

Thm: A TM (maybe multiple tapes, maybe nondeterministic) can simulate a real computer.

Proof: Assume that our "real" computer is a random-access stored-program computer (Von Neumann architecture).

We simulate it with a TM;
if the computer takes n steps to perform some operation, TM takes n^6 steps.

Sketch:



Our TM also needs to be able to do the things an assembly language can do:

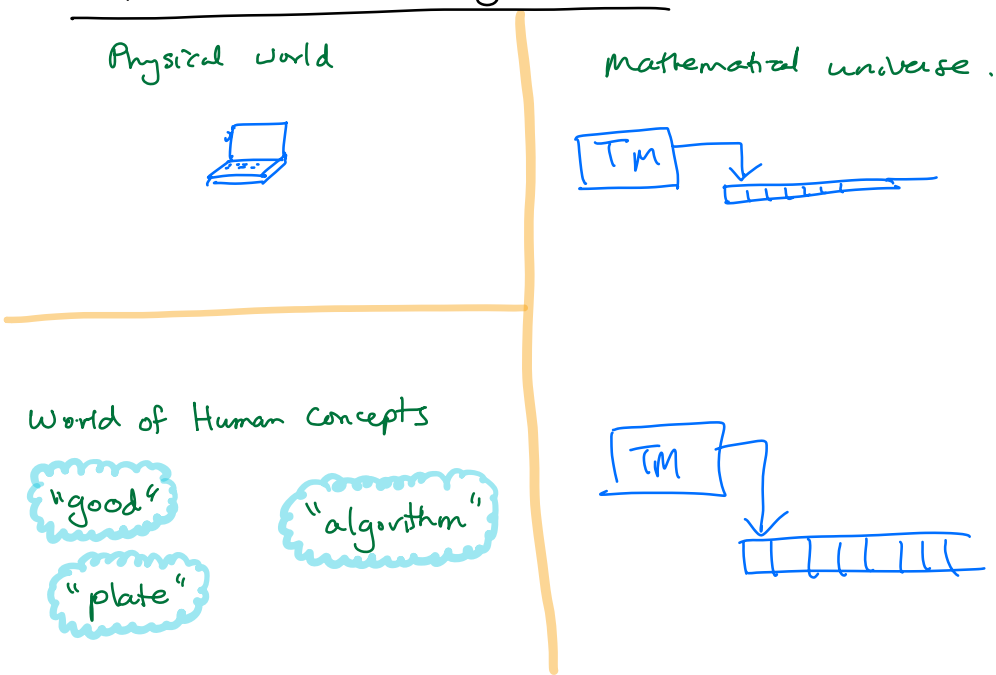
Add / Subtract / OR / AND (ALU stuff)

Load, Store, etc (Memory access stuff)

We can devise a TM to do these things.

▨

3.3 Definition of an algorithm.



Church-Turing Thesis:

∃ an algorithm to compute something ⟺ (if and only if) ∃ a TM that computes it

"TMs" and "algorithms" exist in different epistemological realms.

No mathematical theorem will equate them.

But the Church-Turing thesis says,

"TM captures mathematically what we mean when we say 'there's an algorithm for that'"