COMS W3261 — Lecture 9: Deciding, Recognizing, Enumerating, & Beyond

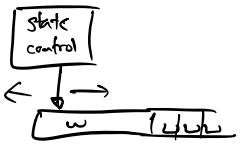
Teaser: Is the language of palindromes over $\{0,1\}$ Turing-recognizable?

$$L = \{ ww^R,\ w0w^R,\ w1w^R, \text{ for any } w \in \{0,1\}^* \}$$

(Implementation-level description.)

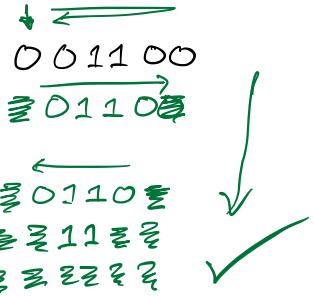
M: "On input w, we

(0. Accept if the tape has ε or σ^* symbols on the tape.)



1. "Remember" the leftmost symbol and erase it.
   - ↳ 0 or 1-representing state
   - ↳ rewrite with blank $\sqcup$.
2. Traverse the tape left to right and check if the rightmost symbol matches what we remember.
   - ↳ If not, reject.
   - ↳ If yes, erase the rightmost symbol, go back to the left end of the input, and repeat from step 0.

Example:



Announcements for today: HW #5 due Monday, 8/2/21 @ 11:59 PM EST

Final info posted on CW/Ed.

Readings: Sipser 3.1 (TMs)

Sipser 3.2 (Turing Machine Variants)

Sipser 3.3 (Church-Turing thesis, TM → algorithm)

Today: 1. Review
2. Variant TMs: TMs with gadgets / superpowers.
3. From TMs to "algorithms"

Example. Multiplication on TMs.

Goal: Decide the language $C = \{a^i b^j c^k \mid ij = k \text{ and } i,j,k \geq 1\}$.

Idea here: for each a, cross off b c's.

M_3 = "On input string w: $\Gamma = \{a, b, c, \cancel{a}, \cancel{b}, \cancel{c}, \sqcup\}$

1. Scan input left to right to ensure we have some string in $a^+b^+c^+$, reject if not. (Easy - like a DFA.)
2. Return to left.
3. Cross off one a:
   Shuttle between the b's and the c's, crossing off one c per b.
   (Reject if we run out of c's.)
4. If there is a remaining uncrossed a, uncross all of our b's and return to step 2.
5. Accept if we cross off the last a, b, and c on the same round.
   If there are c's leftover, reject.

Input aabbbcccccc = ✓

aabbbcccccc = accept.

for each a:
  for each b:
    cross off a c;

a	c	c	c
a	c	c	c
a	c	c	c
	b	b	b

$ij = jc$

How do we "uncross" something?

$a \longrightarrow \cancel{a} \longrightarrow a$ $\quad \{a, \cancel{a}\} \in \Gamma$

$\delta(q, a) \longrightarrow \delta(\cdot, \cancel{a}, \text{move})$

$\delta(q, \cancel{a}) \longrightarrow \delta(\cdot, a, \cdot)$ $\quad a \longrightarrow \dot{a}$

Example. Element Distinctness.

Goal: Decide $E = \{\#x_1 \#x_2 \# \cdots \#x_\ell \mid$ Each $x_i \in \{0,1\}^*$ and $x_i \neq x_j$ for $i \neq j\}$

Idea: Compare x_1 with all x_i, $i > 1$. Then compare x_2 with x_i, $i > 2$. And so on.

TM M_3 = "On input string w:

0. Reject if the input is not in the correct form. Accept if there is only one input.

1. Mark the first two # with a dot like this:

$\dot{\#} x_1 \dot{\#} x_2 \# x_3 \cdots$

2. Scan the two binary strings to the right of the marked # by zig-zagging back and forth, and reject if they match exactly.

$\# \cancel{0}\cancel{0}11 \; \dot{\#} \cancel{0}\cancel{1}0\dot{1}$

3. If possible, move the right mark to the next # and repeat step 2.

If the right mark is at the last # sign, move the left mark forward and the right mark back next to the left mark.

If the two marks are on the last two #, accept.

Takeaway: TMs can solve 'higher-level' recognition problems

than previous automata — can loop and store information freely.

Break: 5 min. Back at 11:02.

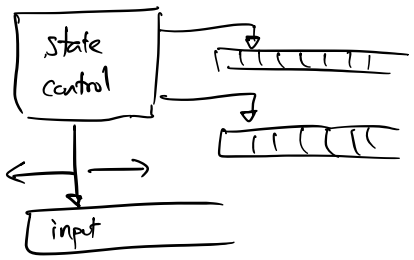
---

2. Variant Turing Machines

2.1 - Multitape TMs.

Idea: TMs with more than one head, random-access tape.

one head per tape.



k tapes:
one with input,
k-1 blank.

Formally: Like a single-tape TM, with the transition function defined as follows:

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

Q: state; Γ^k: input symbols read; Γ^k: write symbols

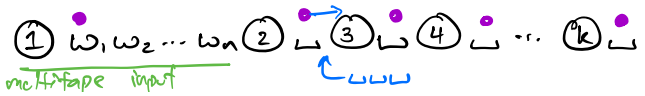
For instance: $\delta(q_i, a_1, a_2, \cdots, a_k) = (q_j, b_1, \cdots, b_k, L, L, R, S, R \cdots)$

Theorem: Every Multitape TM has an equivalent single-tape TM.

Proof: Simulate all k tapes of a given k-tape TM on a single tape. Given a multitape TM M_{multi}, define the following equivalent TM:

M = "On input w:

(1) Set up the input tape to look like k tapes of M_{multi}.



(2) Mark the virtual "tape heads" purple.

(3) Simulate M_{multi} using the transition function. (We have 1 tape head, but we simulate k tape heads by moving our marks around.

(4) If we run out of space on a virtual tape, run a special subroutine to shift the whole tape contents over and make space.

(5) Accept/reject if our simulation accepts/rejects." ∎

Takeaway: To show some language is Turing-recognizable or decidable, we can assume our TMs have multiple tapes w/o loss of generality.

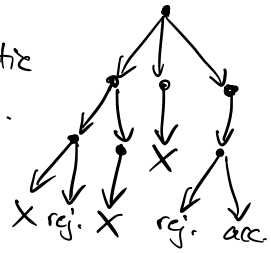
2.2 Nondeterministic TMs.

Why not?

Formally: Same 7-tuple as a deterministic TM, but a difference in the transition function.

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

- Accept if any branch of computation accepts, reject if all reject (or dic.)



nondeterministic computation.

Theorem: Every nondeterministic TM has an equivalent deterministic TM.

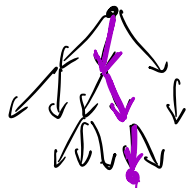
Proof sketch. Every nondeterministic computation looks like a tree. We'll build a TM to simulate an NTM by searching this tree for an accept state. We'll use BFS (breadth-first search) and accept if any branch reaches an accept state.

(Why not DFS (depth-first search) in this case?)

(Worry: infinite loops → infinite branches.)

To traverse the tree, we'll use a TM D with three tapes.

- input tape will store the input to our NTM unchanged.
- address tape tracks our location in the computation tree.



address: 221. Presupposes some order over nondeterministic choices.

- work tape to simulate computation.

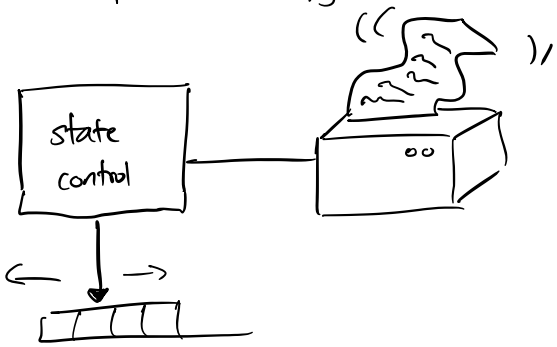
On input w, D will repeatedly increment the address according to BFS, simulate the machine up to this address by copying the original input onto the work tape and making decisions according to the address, accept if we are in an accept configuration.

If the BFS terminates, we reject. Thus our simulation accepts/rejects if and only if the original NTM accepts/rejects. ∎

Takeaway: To prove languages are Turing-recognizable or decidable, we can assume nondeterminism without loss of generality.

2.3 Enumerators

Idea: what happens if you give a TM a printer? We'll imagine such a TM that can compute and copy the input tape to its printer at any time.



Theorem: A language is Turing-recognizable if and only if some enumerator enumerates it. writing down all strings (possibly infinite process.)

Example: Enumerate 0^*.

Enumerator that prints $\varepsilon, 0, 00, 000, \ldots$.

Proof idea: Convert TM $\leftrightarrow$ Enumerator.

1. If some enumerator E enumerates a language, some TM M recognizes it. Define M as follows:

> M = "On input w:
>
> Run E. Every time E outputs a string, check that string against w and accept on a match."

2. If some Turing Machine M recognizes a language, there exists some enumerator E that enumerates it. Let $s_1, s_2, \ldots$, be a list of all strings over the alphabet Σ of M.

(Example. $\Sigma = \{0,1\}$, $s_1 = \varepsilon$, $s_2 = 0$, $s_3 = 1$, $s_4 = 00$, $s_5 = 01, \ldots$).

E will run as follows.

E = "On input w,

Repeat the following for $i = 1, 2, 3, \ldots$

Run (simulate) M for i steps of computation on $s_1, s_2, \ldots s_i$.

If any computations accept, print out the corresponding string s_j."

Why will E eventually output every string in $L(M)$? Suppose $s_j \in L(M)$, and M accepts s_j after k steps. Then, E will print s_j on iteration $\max(j,k)$ of the loop.

(Why not run on s_1 until we finish, then s_2, then $s_3 \ldots$?)

(Why not run on each s_i in turn for some large number of steps?)

(issue: we assume M recognizes $L(M)$, but not that it decides. So on $s_i \notin L(M)$, it might run forever.)

Note: The Turing-recognizable languages are often called the Recursively Enumerable languages (the class RE) because of this theorem.

(MIP* = RE.)

Break — back at 12:15.

3. From TMs to "general-purpose algorithms."

We've seen that several additional powers don't increase the set of languages that TMs recognize.

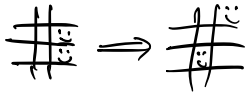
- Multiple tapes
- Nondeterminism.

Many other powerful automata have been proposed equal in power to TMs.

- Post-Turing Machines (Post, 1936.)
- Lamda calculus (Church, 1930s.)
- Automaton with a queue → [| | |] →.
- Wang Tile (Hao Wang, 1961) 

// more in Sipser 3.3

Anything that can simulate a TM can be used to recognize & decide languages. (This property is called Turing-completeness.) Examples:

- Most programming languages (C, Java, Python...)
- LaTeX.
- Conway's Game of Life. 
- MS Excel, Powerpoint
- Minecraft, Portal, Magic the Gathering.

The Church-Turing thesis: "our intuitive notion of a fully specified process for performing a task corresponds precisely to what a TM can do."

Encoding Problems.

Idea: we want to describe TMs at a high level. We also want to maintain formal correctness — we want to be able to reduce to a formal 7-tuple in principle.

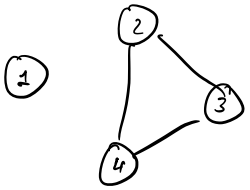
Description levels:

- Formal descriptions of TMs (7-tuples)
- Implentation-level descriptions (Describe how we move the head, manage the tape.)
- High-level descriptions: Precise prose that describes an algorithm while ignoring implementation details.

Observation: All finite mathematical objects can be encoded as strings of symbols.

Example:

G:



$G = (V, E)$

Define an encoding, marked by angle brackets $\langle\rangle$, which represents a graph as "{vertices}{edges}"

So $\langle G \rangle$ = "{1,2,3,4}{(2,3)(2,4)(3,4)}"

Encodings free us up to think about manipulating graphs, not strings.
// think: compiler.

Note: If we see $\langle O \rangle$ as a TM input, we will assume our TM checks and rejects bad input.

Next time: show many languages are decidable, recognizable, and sometimes not decidable/not recognizable.

Reading: Sipser 3.1, 3.2, 3.3.