Oberlin College
CSCI 383
Fall 2026
Lecture 22. Equivalent Models of Computation
Kozen - Section 30

Last time: TMs can count, record things, and even do simple math, if you're careful.

Today. Recall:

- Recursive: Accepted by a total TM.
- Recursively enumerable: Accepted by a TM.

Asked some questions:

1. Recursive $\rightarrow$ Recursively enumerable. Is the containment strict?
2. What pieces of our model for Turing machines are important?
3. Closure properties?

Today: Answering number 2, some. And thinking a little about 1.

Multiple Tapes

Homework 7: Two stacks is better than one.

How about two tapes?

A two-tape Turing machine works as follows:

- Input written on first tape.
- Two read/write heads.
- Transitions: $\delta : Q \times \Gamma \times \Gamma \rightarrow Q \times \Gamma \times \Gamma \times \{L, R\} \times \{L, R\}$

  (Each transition reads/writes with each head, and moves each head left/right.)

How do you prove one model as at least as powerful as another? (Simulate the latter w/ the former.)

Is there anything we can say off the bat? (Can simulate one tape with two.)

We'll prove they're equivalent. (i.e. simulate two-tape machine with one)

Problem 1: Need to simulate two tapes.

- For DFAs, simulating two state spaces: annoying.
- For NPDAs, simulating state and stack simultaneously: annoying.

To simulate two tapes: write contents of the first, then the second.

Problem 2: Need to simulate two read/write heads.

Can we store the location in state? No.

Mark location on tape.

Algorithm. To simulate a step of two-tape machine:

- Sweep through first tape, simulate step.
- If you need more room, recopy contents of second tape.
- Sweep through second tape, simulate step.

How would you modify this apporoach for k tapes?

Note: Also gives us a simulation for two-way infinite tape.

Enumeration Machines

Definition: An enumeration machine is a Turing machine with the following modifications:

- Two tapes (enumeration tape and work tape)
- No input
- Special enumeration state e. On entering e:
  - Contents of enumeration tape recorded.
  - Enumeration tape cleared.
- $L(M)$ is the set of strings that will be enumerated by M

Regular Turing machine: tests primality. Enumeration machine: prints all primes.

Questions:

1. Given an EM M, can we construct a TM for $L(M)$?
2. Given an EM M, can we construct a total TM for $L(M)$?
3. Given a total TM M, can we construct an EM for $L(M)$?
4. Given a TM M, can we construct an EM for $L(M)$?

Which of the first two are harder? Which of the second two are harder?

Given EM M, our TM:

- Stores input on a tape.
- Simulates M on two more tapes.
- Whenever M enumerates, checks if x was the enumerated string.

Total TMs

Assume we're given a total TM M. Goal: construct a machine that enumerates $L(M)$.

Idea: Try M on every string.

Outside loop: List every string $x \in \Sigma^*$. Keep a separate tape for this.

- ϵ
- 0
- 1
- 00
- 01

- etc.

Inside procedure: Run M on x.

- Store M's tape on a separate tape.
- Store M's state in finite control.

On accept: copy to enumeration tape, enumerate.

How does this break if M is not total?

Running M in sequence requires M to halt on all inputs.

Timesharing

Idea: Run many instances of M in parallel:

- Run one step of M on ϵ
- Run one step of M on ϵ and one step on 0
- Run one step of M on ϵ, 0, and 1

Whenever M accepts x, enumerate x.

How can we do this with a Turing machine.

First tape: enumerate all $x \in \Sigma^*$.

For each $x \in \Sigma^*$:

- Perform one step of all existing runs.
- If any runs accept, enumerate.
- Start a run of M on x after the existing runs.

Can we store the state for all these runs in the finite control?

Solution: Store state in each row with tape contents.

For any string x and step k, machine will run step k of M on x.

Hence, all strings accepted by M will be enumerated at some point.

$\Rightarrow$ machine enumerates $L(M)$.

Conclusion: A language is RE if and only if it has an EM.