## Week6 friday

To define a Turing machine, we could give a

- Formal definition: the 7-tuple of parameters including set of states, input alphabet, tape alphabet, transition function, start state, accept state, and reject state; or,
- Implementation-level definition: English prose that describes the Turing machine head movements relative to contents of tape, and conditions for accepting / rejecting based on those contents.
- **High-level description**: description of algorithm (precise sequence of instructions), without implementation details of machine. As part of this description, can "call" and run another TM as a subroutine.

**Theorem 3.21** A language is Turing-recognizable iff some enumerator enumerates it.

## **Proof**:

Assume L is enumerated by some enumerator, E, so L = L(E). We'll use E in a subroutine within a high-level description of a new Turing machine that we will build to recognize L.

**Goal**: build Turing machine  $M_E$  with  $L(M_E) = L(E)$ .

Define  $M_E$  as follows:  $M_E$  = "On input w,

- 1. Run E. For each string x printed by E.
- 2. Check if x = w. If so, accept (and halt); otherwise, continue."

Assume L is Turing-recognizable and there is a Turing machine M with L = L(M). We'll use M in a subroutine within a high-level description of an enumerator that we will build to enumerate L.

**Goal**: build enumerator  $E_M$  with  $L(E_M) = L(M)$ .

**Idea**: check each string in turn to see if it is in L.

How? Run computation of M on each string. But: need to be careful about computations that don't halt.

Recall String order for 
$$\Sigma = \{0, 1\}$$
:  $s_1 = \varepsilon$ ,  $s_2 = 0$ ,  $s_3 = 1$ ,  $s_4 = 00$ ,  $s_5 = 01$ ,  $s_6 = 10$ ,  $s_7 = 11$ ,  $s_8 = 000$ , ...

Define  $E_M$  as follows:  $E_M =$  " ignore any input. Repeat the following for i = 1, 2, 3, ...

- 1. Run the computations of M on  $s_1, s_2, \ldots, s_i$  for (at most) i steps each
- 2. For each of these i computations that accept during the (at most) i steps, print out the accepted string."

## Nondeterministic Turing machine

At any point in the computation, the nondeterministic machine may proceed according to several possibilities:  $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$  where

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

The computation of a nondeterministic Turing machine is a tree with branching when the next step of the computation has multiple possibilities. A nondeterministic Turing machine accepts a string exactly when some branch of the computation tree enters the accept state.

Given a nondeterministic machine, we can use a 3-tape Turing machine to simulate it by doing a breadth-first search of computation tree: one tape is "read-only" input tape, one tape simulates the tape of the nondeterministic computation, and one tape tracks nondeterministic branching. Sipser page 178

Two models of computation are called **equally expressive** when every language recognizable with the first model is recognizable with the second, and vice versa.

**Church-Turing Thesis** (Sipser p. 183): The informal notion of algorithm is formalized completely and correctly by the formal definition of a Turing machine. In other words: all reasonably expressive models of computation are equally expressive with the standard Turing machine.

Claim: If two languages (over a fixed alphabet $\Sigma$ ) are Turing-recognizable, then their union is as well.
Proof using Turing machines:
Proof using nondeterministic Turing machines:
Proof using enumerators:
1 1001 using chamerators.