The Church-Turing Thesis

We gave an argument, nearly a proof on the equivalence between a formal model of computation and the ~~equivalent~~ intuitive notion of computation. Lets restate this in the notation used in our class so far. Let $\mathcal{L}(Human)$ be the class of languages which correspond to those which have algorithms, in the intuitive sense. Let $\mathcal{L}_D(TM)$ correspond to the class of languages which have Turing machines to decide them.

$\mathcal{L}_D(TM) \subseteq \mathcal{L}(Human)$. The proof is obvious. You have the ability to fathom a Turing machine. Given a TM M and its input w you may simulate M on w in your mind.

$\mathcal{L}(Human) \subseteq \mathcal{L}_D(TM)$ We took a model of a human performing the action of computation, distilled it only to its barest essentials, and argued there must exist a corresponding Turing machine. Turing's thesis is this simulation.

We may then conclude $\mathcal{L}(Human) = \mathcal{L}_D(TM)$. Proofs involving certain Turing machines, existence or non-existence of therefore implicate the intuitive notion. Turing's more classical demonstration of the Turing machine as the ultimate computer. A TM only has three pathetic instructions. We give several obvious generalizations C such that $\mathcal{L}_D(TM) \subseteq \mathcal{L}(C)$ but then show $\mathcal{L}(C) \subseteq \mathcal{L}_D(TM)$, to show they cannot be strictly more powerful. Only equal. This should be taken as evidence of the Church-Turing thesis, which we formalize below. For any kind of computational model, decision procedure, mechanical process C

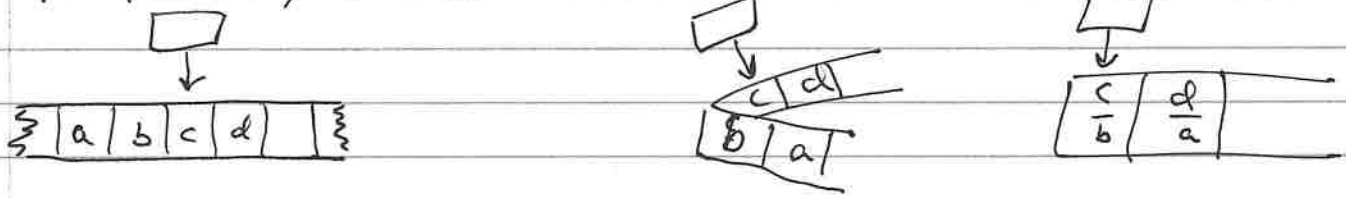
$$\forall C \quad \mathcal{L}(C) \subseteq \mathcal{L}(TM)$$

or equivalently

$$\nexists C \quad \mathcal{L}(TM) \subsetneq \mathcal{L}(C).$$

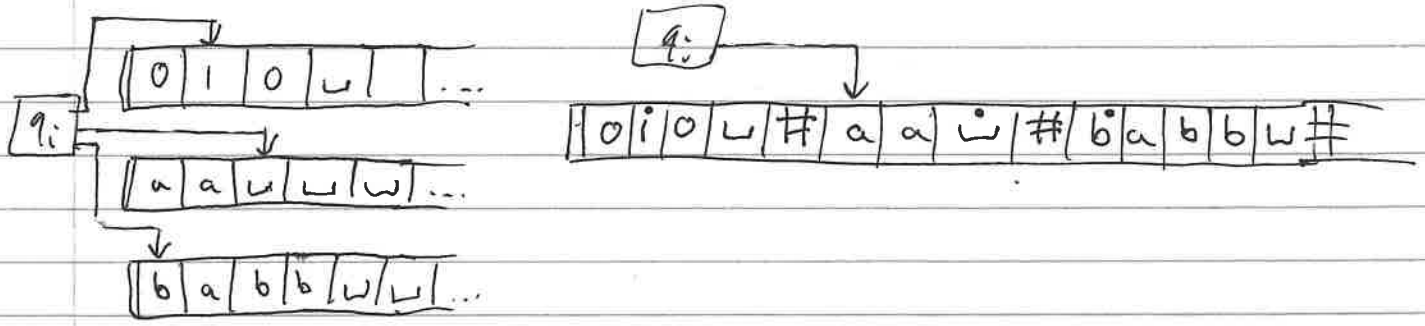
stay TM First suppose we generalized the definition of a Turing machine to allow it to stay put, instead of forcing it to move L or R. Call this kind of machine stay TM with transition function $\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$. Note every normal TM without stay, is also one with. So $\mathcal{L}(TM) \subseteq \mathcal{L}(\text{stay TM})$. We prove $\mathcal{L}(\text{stay TM}) \subseteq \mathcal{L}(TM)$ by simulating a stay TM on a normal Turing machine. If a stay TM has a normal instruction, keep it the same. If it has a stay instruction of the form $a \to b, S$ convert it to the following two instructions $a \to b, R$, $\begin{matrix} a \to b \\ b \to b \\ \sqcup \to b \end{matrix}, L$. moving right then moving back left immediately is equivalent to staying, so $\mathcal{L}(\text{stay TM}) \subseteq \mathcal{L}(TM)$.

2way TM The definition of the Turing machine has a one way infinite tape. What if it was two way infinite? Certainly $\mathcal{L}(TM) \subseteq \mathcal{L}(\text{2way TM})$. We show this generalization gives no power. We simulate the two way tape on a one way tape by folding the tape in half and increasing the tape alphabet quadratically.



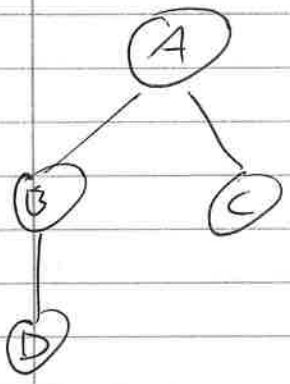
If it tries to move left off the tape, the instructions are flipped and it begins modifying only the denominator. $\Gamma^2 = \{a/b \mid a, b \in \Gamma\}$ we see we can simulate a 2way tape on a one way one so $\mathcal{L}(\text{2way TM}) \subseteq \mathcal{L}(TM)$.

A bidirectional tape is kind of like two tapes. A multitape TM has some (finite) number of tapes k with transition function $\delta: Q \times \Gamma^k \to \Gamma^k \times Q \times \{L, R\}^k$. Certainly $\mathcal{L}(TM) \subseteq \mathcal{L}(kTM)$ by ignoring $k-1$ tapes. We prove $\mathcal{L}(kTM) \subseteq \mathcal{L}(TM)$



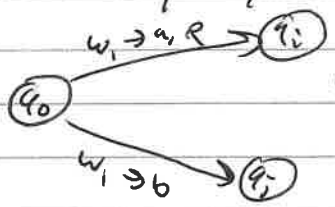
The simulation proceeds as follows. for each step of the kTM, we scan through our linear type we update the symbols near the dotted ones accordingly. If we ever read more ~~type~~ blanks for any type, we pause the simulation, enter a subroutine which shifts the elements on the type and inserts a blank, simulation is then resumed. Its really slow, but correct. We can conclude $\mathcal{L}(kTM) \subseteq \mathcal{L}(TM)$.

Lets do one last generalization, a big one. Define an NTM, a nondeterministic Turing machine to be like a normal one except δ is not deterministic. $\delta(q_i, a)$ is a set and at one valid configuration, there may exist multiple succeeding ones. If you consider computation like a tree, a deterministic machine has arity 1, and is then a line. A nondeterministic one is more like a tree, with perhaps infinitely long branches. Since the transition function is like $\delta: Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$, there are multiple next configurations. We will attempt to deterministically explore this tree to find an accepting configuration. DFS runs into an immediate problem. We may get in a loop down one wrong computation path, so the trick is to use BFS



$[A] \to [BC] \to [CD] \to [D] \to [\,]$

pop a node off the queue, and push its children.



$\# q_0 w_1 \ldots w_n \# \sqcup \sqcup \ldots$

$\# \sqcup \sqcup \ldots \sqcup \# \# q_0 q_i w_2 \ldots w_n \sqcup \# b q_i w_2 \ldots w_n \sqcup \#$

append configurations and stop when one accepts. We see then that $\mathcal{L}(NTM) \subseteq \mathcal{L}(TM)$

All these generalizations of Turing machines failed to increase the power, since for each, we were able to simulate them on a normal Turing machine. We see thus that the pathetic 3 instruction machine is not so weak as it first appears, and $\mathcal{L}(TM)$ is a large and reasonable class.

One thing about these models (except the NTM) is they all seem to be physically realizable. They seem realistic. They are bound by our physical laws in the following ways.

1) A program descriptions are of finite length

2) Finite work is done in finite time.

It is actually easy to come up with machines which are Super-Turing, but these are all also obviously unrealizable. For example, they can compare arbitrarily long strings in constant time, when it takes at linear time just to look at the input.