CS 4510 Automata and Complexity October 4th 2023

Lecture 10: Church-Turing Thesis

Lecturer: Abrahim Ladha *Scribe(s): Rishabh Singhal*

1 Introduction

The Church-Turing Thesis cannot be proved. Most agree that it is some kind of definition or worse, a "working hypothesis". Here, we will give the closest thing to a proof possible. Recall the narrative of our class. First we did regular languages, that was level one. We pumped out of those to get some context-free languages, that was level two. Now we are on level three, Turing machines. The Church-Turing Thesis asserts that there is no level four. The Church-Turing Thesis has two parts:

Turing's Thesis : The Turing Machine is equivalent in power to the Human Mind

Church's Thesis : Any serious formalization of computation is equivalent to the Turing machine

Together, these imply that a Turing Machine, although incredibly simple, is an excellent choice for us to reason about computation. The intuitive notion of computation corresponds to the Turing machine, and that the intuitive notion of computation may be considered independent of a specific formalism. Both Turing's Thesis and Church's Thesis are required.

A philosophical argument is unlike a proof. We want to make a convincing argument of some statement. The way we will do so is make atomic jumps, each convincing, then argue that the composition must be convincing.

2 The Direct Appeal to Intuition

These notes have been typeset separately: https://ladha.me/files/fancy-turing-notes.pdf

3 Church Turing Thesis

We now give our statement of the Church-Turing Thesis. For any kind of fathomable computation model, mechanical process, decision procedure C,

$$\forall C \ \mathscr{L}(C) \subseteq \mathscr{L}(TM)$$

Its useful to rephrase this as

$$\nexists C \ \mathscr{L}(TM) \subsetneq \mathscr{L}(C)$$

In human words, the Turing machine is the ultimate computer. The entire complexity and confusion of the mind, at least in our framework of decision problems, can be simplified to the humble Turing machine. This pathetic three button typewriter is equivalent in power to any realizable computer, and there is no greater. It is the supreme.

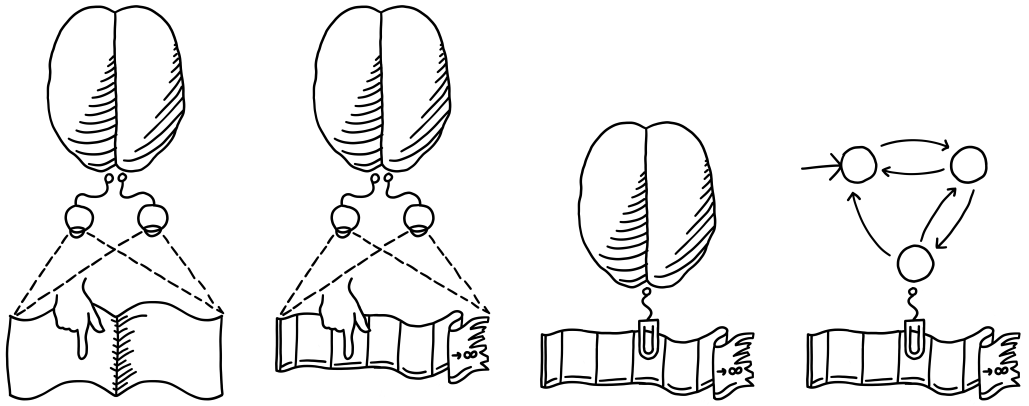


Figure 1: Successive simplications of computation as made in the Direct Appeal to Intuition

4 Evidence

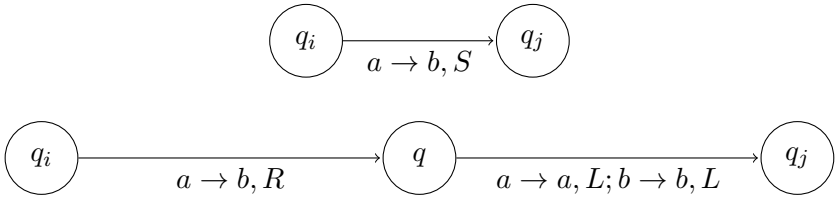
We will now show a more traditional argument in favor of the Church-Turing Thesis. We will attempt to generalize the Turing machine. We will show that each generalization fails to be strictly stronger and is just equivalent.

4.1 Turing machine with Stay

Consider Turing machines with a stay instruction. Instead of moving either left or right, we allow the tape head to remain on the same cell. Its transition function would be defined like:

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$$

It is obvious that $\mathscr{L}(TM) \subseteq \mathscr{L}(stayTM)$. Let's prove $\mathscr{L}(stayTM) \subseteq \mathscr{L}(TM)$. If a Turing machine with stay has a normal L, R move, we leave it alone. If a Turing machine with stay has a S move, we can simulate it as a sequence of L, R moves. Specifically we will choose to move right, then back left[1].

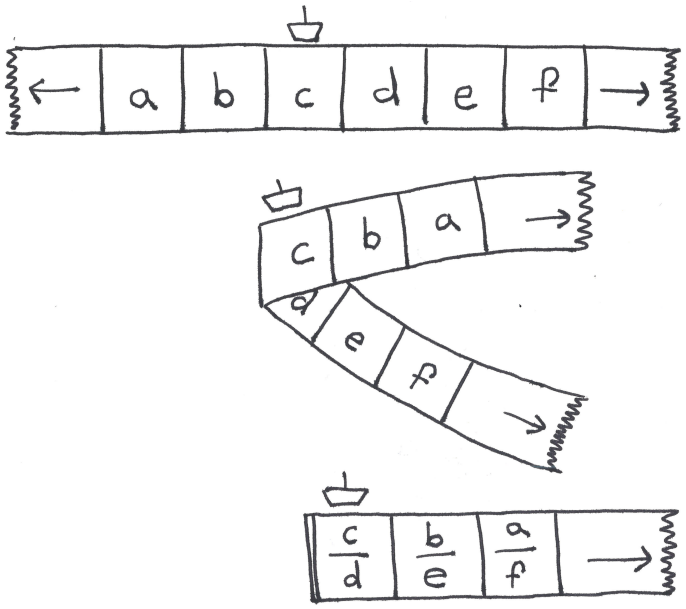


4.2 Turing machine with Two Way Tape

Our Turing machine's tape is only infinite in one direction, so lets generalize it to be infinite in two. This can be called a bidirectional, doubly infinite, or two way tape.

It is true that $\mathscr{L}(TM) \subseteq \mathscr{L}(2wayTM)$, but the simulation needs an extra sentence. We put our one way tape on the two way tape and add a special marker leftmost of our

[1]The only reason we don't move left then right is the case we are on the leftmost square.



tape. If we ever read it, we force ourselves right. Now lets show $\mathscr{L}(2wayTM) \subseteq \mathscr{L}(TM)$. There were many excellent simulation suggestions in class, but the elegant one is to just fold the tape in half! Here we extend our tape alphabet to cover pairs as so.

$$\Gamma^2 = \left\{ \frac{a}{b} \mid a, b, \in \Gamma \right\}$$

If you are in the right half of the tape, the transition function will essentially ignore the bottom half of the symbol. If we attempt to move into the left half, we start ignoring the tops and looking at the bottoms. We also flip every L, R move.

4.3 Multi-Tape Turing Machine

If you think about it, the negative half of a bidirectional tape is like a second tape. Lets define a Turing machine with multiple tapes. A multi-tape Turing machine is just that. It has k tapes it may read, right and move on all independently. Its transition function would look like

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

At some state, it will read the position of its k tape heads, makes k writes, and moves each head either left or right independently. Certainly $\mathscr{L}(TM) \subseteq \mathscr{L}(kTM)$ by simply ignoring all tapes except the first one.

We want to show $\mathscr{L}(kTM) \subseteq \mathscr{L}(TM)$. In order to do so, we are going to simulate the k tapes on a single tape. We initialize our single tape as

$$\#\dot{w}_1 ... w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \dot{\sqcup} \# \dot{\sqcup} \# ... \# \dot{\sqcup} \#$$

The dot represents the position of that tape head over its tape. As we simulate a read, write, move of each tape head, we will scan over our tape making the appropriate adjustments.

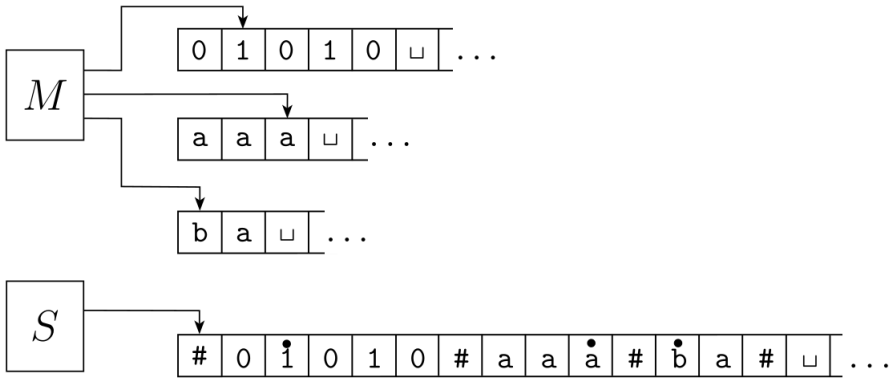


Figure 2: A Three tape Turing machine being simulated on a single tape, from the Sipser book

If we need more space than allocated, we pause the simulation, enter a shifting subroutine, insert blanks appropriately, and then continue. What was essential for this simulation was that at any point in time, only a finite amount of space has been used. A Turing machine cannot use the infinite nature of the tape in any useful way.

4.4 Nondeterministic Turing machine

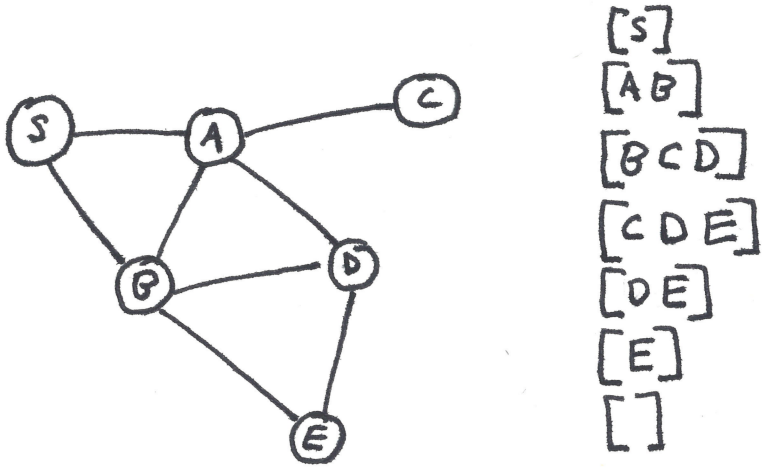
A nondeterministic Turing machine is defined exactly as you might think. At some state reading some symbol, there are more than one outgoing transitions which could be taken. There exists more than one computation path. Its transition function would be defined as

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

Its certainly true that $\mathscr{L}(TM) \subseteq \mathscr{L}(NTM)$ as a nondeterministic Turing machine is simply a generalization of a Turing machine. We now show the reverse way, that $\mathscr{L}(NTM) \subseteq \mathscr{L}(TM)$.

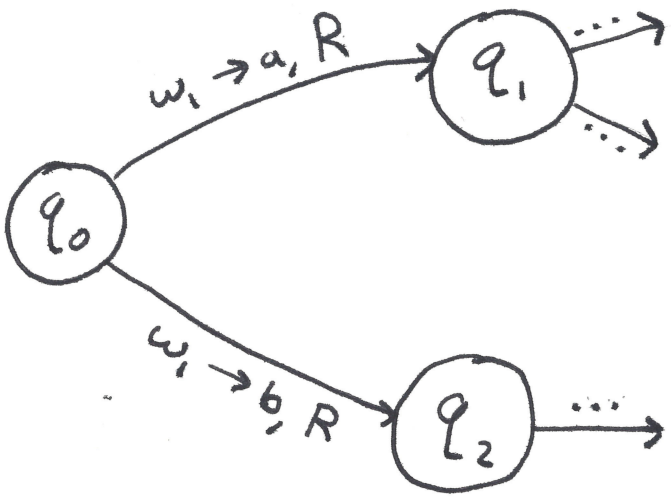
We can view a nondeterministic computation like a rooted tree. Each node in this computation tree can be represented by a configuration, with the initial configuration representing the root, $q_0 w_1 ... w_n$. Our deterministic simulator is going to attempt to search this tree for an accepting computation, if one exists. If our nondeterministic machine has some accepting configuration in this tree, then there exists an accepting computation path and this computation path halts. Our deterministic simulator will find this accepting configuration, and also halt. If our nondeterministic machine has no accepting configuration in this tree, then on all computation paths it either rejects or loops. So our tree may go on forever. Similarly, our deterministic simulator will halt or search the tree forever.

Recall the many tree traversal algorithms. An initial idea is to use DFS: Depth first search. This is not a good idea. If the first computation path we find is some infinite loop, we may miss an easy accepting configuration. The next idea is then to use BFS: Breadth first search. Our deterministic simulator is going to implicitly perform BFS on the computation tree.



Recall how BFS works with a queue. We pop an element off the queue and push its children. It has the unique property that no grandchild is visited before every child is visited.

We are going to use the tape kind of like a queue. We will pop (read, then erase) a configuration off the front of the tape, compute the possible next configurations, and push (write) them to the end of the tape. For example if our nondeterministic machine had the following structure:



Then our tape would transform like:

#	q_0	w_1	w_2	....	w_n	#	␣	␣	␣	␣	␣	␣	␣	␣	␣	␣	␣	$\infty \to$

␣	␣	␣	␣	···	␣	#	a	q_1	w_2	···	w_n	#	b	q_2	w_2	···	w_n	#	$\infty \to$

The configuration $q_0 w_1 ... w_n$ yields two configurations nondeterministically, those being $aq_1 w_2 ... w_n, bq_2 w_2 ... w_n$. We would read the initial configuration on the tape, compute its

two next configurations, and append those on our tape. We would then erase the initial configuration. We would then next look at the first configuration on the tape and repeat. While we are doing this, we check if a configuration contains an accept state, and if it does, we accept. If it contains a reject state, we toss the configuration and continue searching the others. Its worth noting the Sipser book does a slightly different more detailed simulation, still using BFS but with three tapes. I recommend you read it. We have shown that nondeterministic Turing machines are no more powerful than Turing machines. Keep this simulation in mind when we discuss the resource bounded variant of the question: $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$.

We have now constructed several obvious generalizations of the simple Turing machine, yet each of them failed to be strictly more powerful. We were able to simulate each of them on our humble device, implying that it perhaps as more power than at first class. I will say it again. The Turing machine is such an interesting device, the entire rest of the course will be dedicated to its study.