

COSC 312 Turing Machine

Project Collaborators

Joshua D. Dalton (jdalton6) - Main Design/Implementer

Terryl Dodson (tdodson3) - Design collaborator/Writer

Tres James (tjames17) - Design collaborator/Narrative Scribe

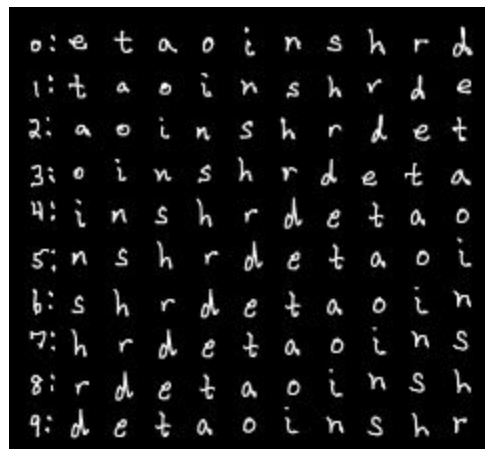
Problem Description/Motivation

Upon first brainstorming for the machine, we went over a few ideas but none of them piqued our interest. However our team member, Joshua Dalton, came across the Wittenstein's family cipher so we all thought it was a good idea to go ahead with the idea of a message cipher. The idea seemed interesting and the goal seemed obtainable within a reasonable amount of time so the group came together on that idea and began implementation. Upon implementation of a baseline machine, we found that a basic message cipher that encoded and decoded one letter or number to a corresponding encrypted letter or number was too simple and we were not sold on whether or not the machine was creative enough for the class, so we scrapped that machine. We mulled over the options for a few days and eventually came up with a new machine design thanks to the quick wits of the group that implemented some machine memory and multiple message shifted encryption keys.

In the actual design of the machine, we were having difficulty moderating the overall complexity so we decided to shrink the alphabet to a smaller size. We use ten letters in our alphabet because that requires a maximum of nine shifts. If we have eleven or more letters, then we'd have the capability of utilizing ten or more shifts. This causes states and transitions to start blowing up, as we have to account

for reading one or two numbers and processing many other letter combinations. With ten letters and nine shifts, we are able to limit ourselves to 103 states and 294 transitions. It consists of the 10 most frequently used letters in the English alphabet: 'e', 't', 'a', 'o', 'i', 'n', 's', 'h', 'r', 'd' (in order of frequency).

Here below is a picture of the alphabetic shift table used in the machine:

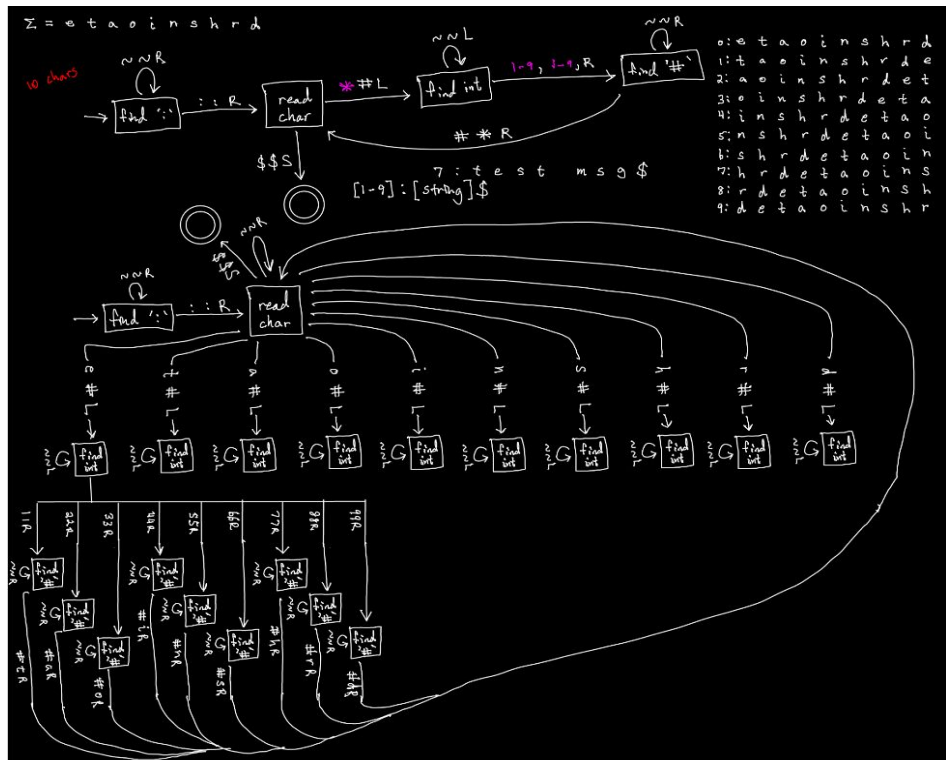


0:	e	t	a	o	i	n	s	h	r	d
1:	t	a	o	i	n	s	h	r	d	e
2:	a	o	i	n	s	h	r	d	e	t
3:	o	i	n	s	h	r	d	e	t	a
4:	i	n	s	h	r	d	e	t	a	o
5:	n	s	h	r	d	e	t	a	o	i
6:	s	h	r	d	e	t	a	o	i	n
7:	h	r	d	e	t	a	o	i	n	s
8:	r	d	e	t	a	o	i	n	s	h
9:	d	e	t	a	o	i	n	s	h	r

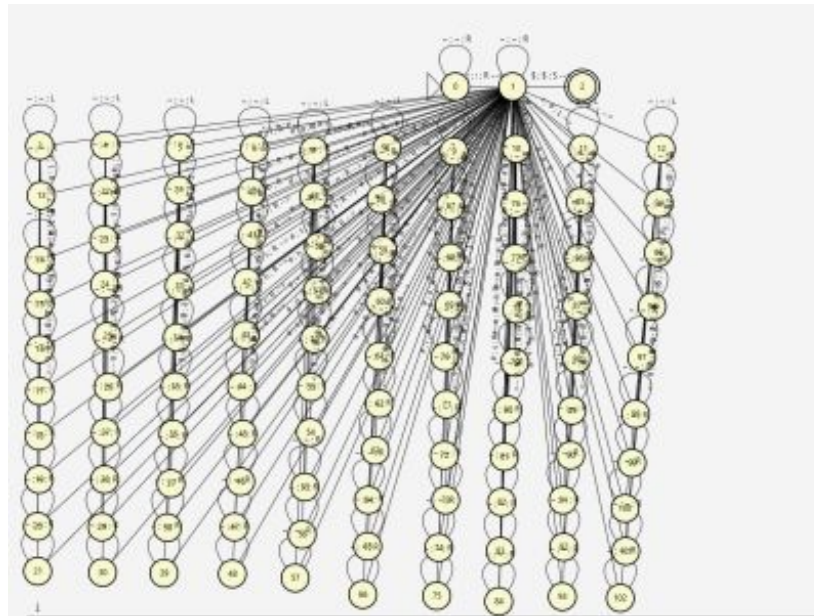
After deciding on this, we proceeded to work on implementation of a final machine. Below, you will find the description and state diagram of our machine.

Functional Description

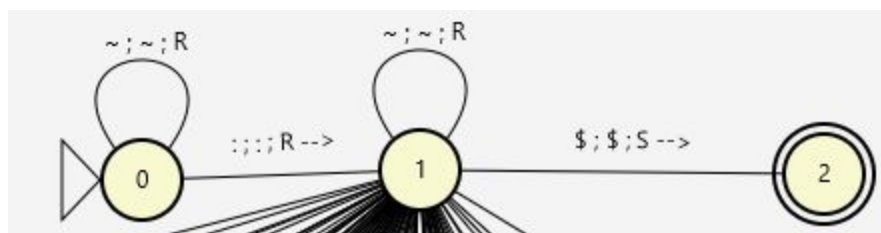
Here is the original design write up that shows the thought process behind the implementation of the machine:



Below you will see how this design corresponds to the states of the machine. This is what the machine looks like when opened up for the first time.



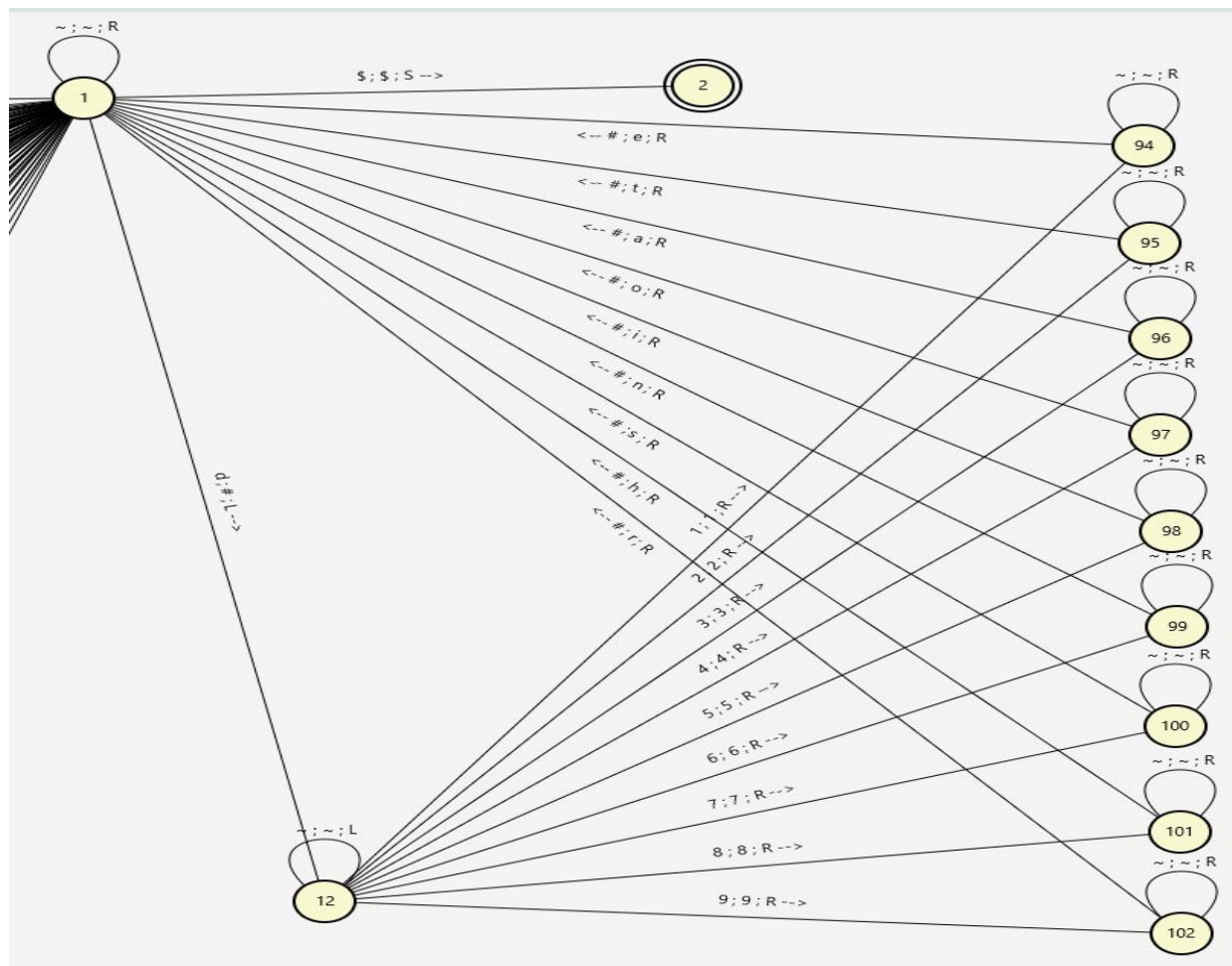
At first glance, it looks like a jumbled mess, but when you break it down it is quite easy to decipher the moving parts. Here below is the starting state (0), main core state (1), and the accept state(2). The machine starts at 0 and does not change state until it encounters the ':' character.



If the next character is a character not in the machine alphabet it is skipped. Otherwise, the machine will overwrite that character with a '#' in order to keep track of where it is in the input string and backtracks to find the shift number (1-9). When the machine finds the '#' it

overwrites it with the new ciphertext letter and transitions back to state one where it is reading in the character that is being encrypted or decrypted. It continues this process until it finds the '\$' which is the accept state (state 2), and the process is finished.

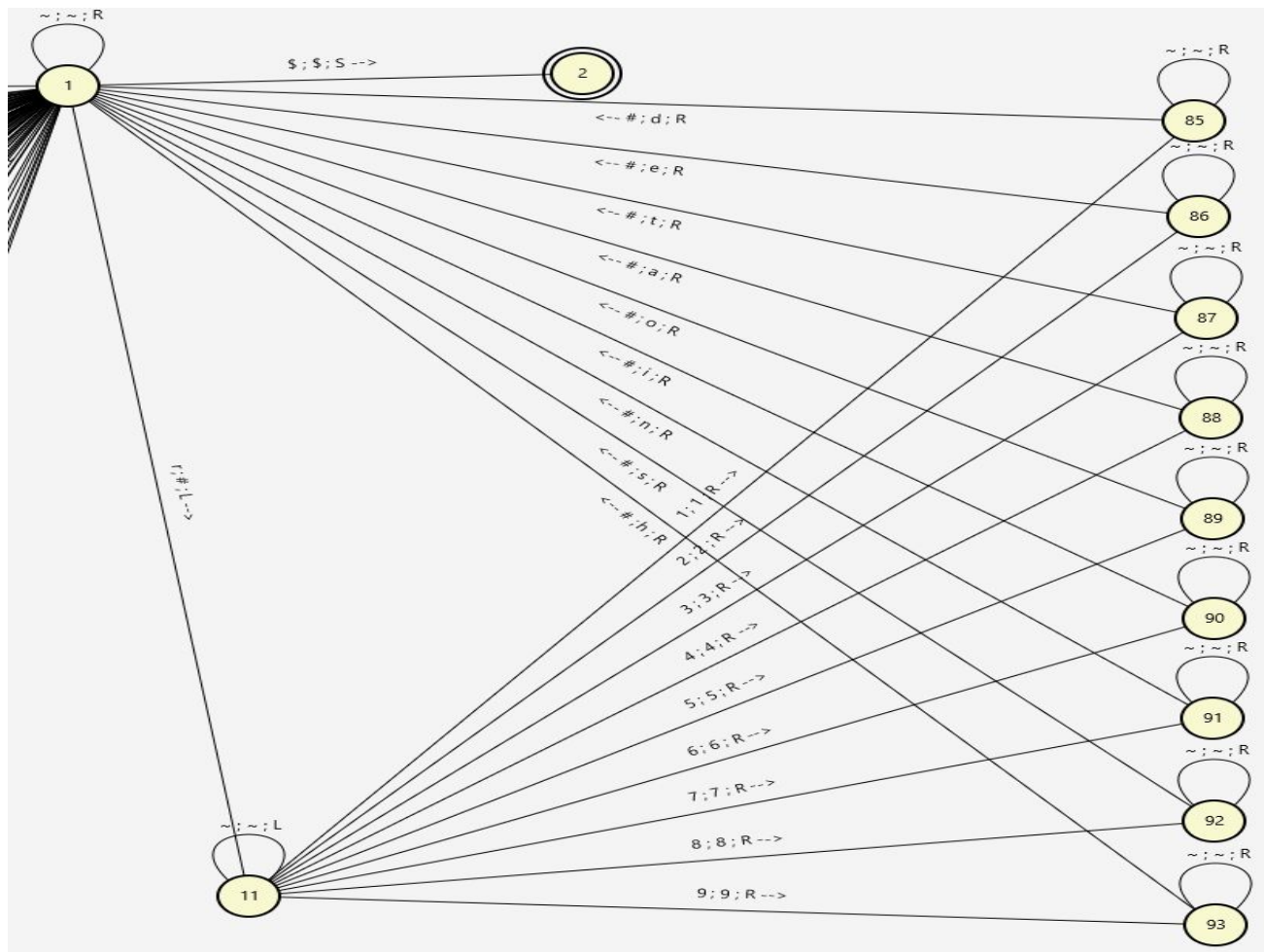
Below, I have moved out a set of states that corresponds to the process I am explaining.



For a step by step explanation of the process, here we have states 1, 12, and 94-102. To start the process, the machine has already found the ':' so it is starting the cipher process. It comes from state one where in this case it found a 'd'. The 'd' is overwritten with a '#' in order to keep track of where the machine is in the string. Then, the machine stays in state 12 until it backtracks (moves left) and finds the shift number. Once it finds the shift number in state 12, it

moves to one based on which shift number it is into states 94-102 (1-9). It reads and writes the shift number back into its initial position. Once it has done that, the machine stays in the state that it's in and moves right in the string until it finds the '#' again. Once this happens, it writes the corresponding cipher letter and goes back to state one. It repeats this process for the letters in the language until it finds the '\$' meaning the input string is complete and it moves into the accept state 2.

Once again I move out a set of states below:



As you can see, states 11 and 85-93 repeat the same process as 12 and 94-102 but for the letter 'r'. I will list below what the rest of the states correspond to as they are the same process as above. You can find them in the whole state diagram above.

- States 3, 13-21 repeat the process for letter 'e'
- States 4, 22-30 repeat the process for letter 't'
- States 5, 31-39 repeat the process for letter 'a'
- States 6, 40-48 repeat the process for letter 'o'
- States 7, 49-57 repeat the process for letter 'i'
- States 8, 58-66 repeat the process for letter 'n'
- States 9, 67-75 repeat the process for letter 's'
- States 10, 76-84 repeat the process for letter 'h'

Input Specification

The input specification for the machine is as follows:

- Proper input is of the form "[shift number: 1-9]:[string]\$"
- The shift number is processed according to this algorithm: x is used to encrypt the string; 10-x is used to decrypt the string. E. g., 3:test\$ -> 3:iodi\$ and 7:iodi\$ -> 7:test\$.
- We have a ten letter alphabet that can be used for the input string: 'e', 't', 'a', 'o', 'i', 'n', 's', 'h', 'r', 'd'
- The machine will allow the user to input other non-numerical symbols and letters but the machine will not change or shift them leaving them as is.
- The machine will not process numeric input because it uses numbers as part of the shifting process and changes the input accordingly.
- Also, '#', ':', and '\$' are not valid members of any input that would be encrypted or decrypted as they are used by the machine to remember the character slot being processed, demarcate the shift number in relation to the input string, and to recognize the end of the input string, respectively.
- The input string **must** end in \$ or it will not halt.