

A Turing Machine simulator written in Haskell

1. Description of what your program does:

My project replicates a Turing Machine made in Haskell. Now a Turing machine was thought of by the mathematician Alan Turing in 1936. A Turing machine can simulate any computer algorithm, no matter how complicated it is. What makes a Turing machine so unique is that it consists of an infinitely-long tape which acts like memory in a computer. At any point in time, the machine has a head which is positioned over the tape, and with this head, the machine can perform various operations.

- Read the symbol
- Edit the symbol
- Move the tape left or right

A Turing machine is a general example of a CPU that controls all data manipulation done by a computer. It is a machine capable of enumerating some subset of strings from an alphabet. Turing machines have a tape of infinite length on which it can perform read and write operations.

2. A description of how your program is organized:

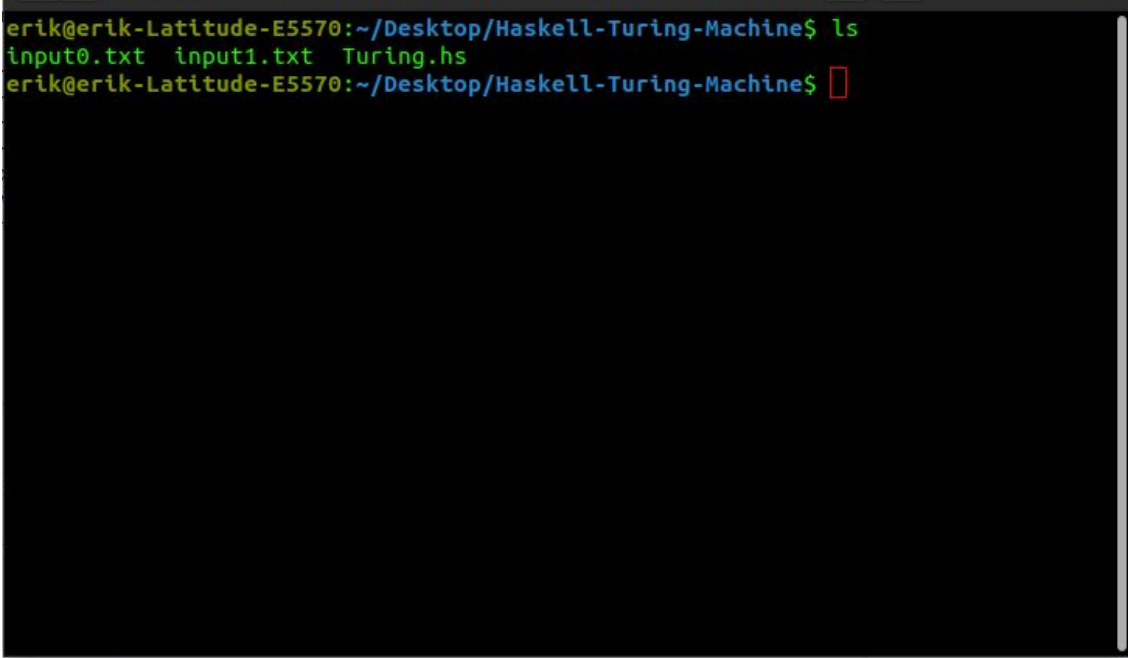
For my project I tried to replicate as best I could a complete Turing Machine. I had started my project hoping it could allow users to input all information at the command line prompt. This proved to be a harder task than I initially realized. So I changed my design to allow the user to input a file. The input file consists of:

- A Tape
- Start state
- Accept State
- Transitions

Once you have a proper input file you either iterate through each step as you work your way through the tape, or you can continue through all the steps in one command. From there the program will decide if the input file will end in an accept state or a trap state.

The following is a guide instruction of how to use my program:

1. Download the zip file and extract the files to your computer.
2. From there open your terminal and navigate to the directory where you have extracted the files.
3. Next you should see all the file within terminal

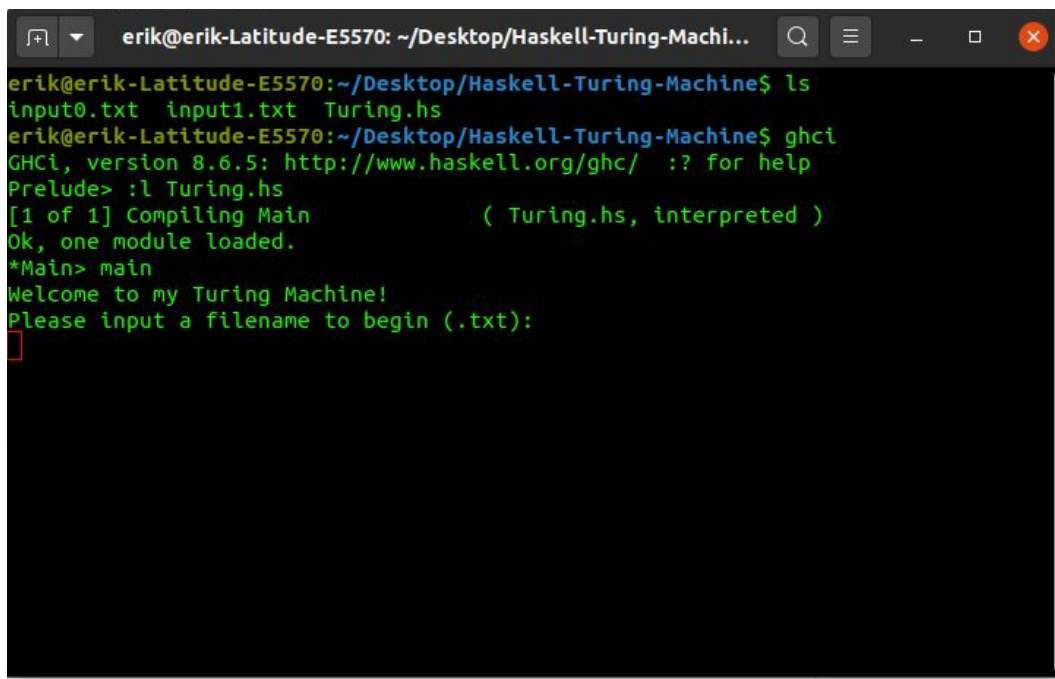
A terminal window with a black background and green text. The prompt is 'erik@erik-Latitude-E5570:~/Desktop/Haskell-Turing-Machine\$'. The command 'ls' has been entered, and the output is 'input0.txt input1.txt Turing.hs'. The prompt is now 'erik@erik-Latitude-E5570:~/Desktop/Haskell-Turing-Machine\$' followed by a red cursor.

```
erik@erik-Latitude-E5570:~/Desktop/Haskell-Turing-Machine$ ls
input0.txt input1.txt Turing.hs
erik@erik-Latitude-E5570:~/Desktop/Haskell-Turing-Machine$
```

4. Type “ghci” into the terminal window, this will then prompt you with the words Prelude>
5. Type :l Turing.hs after the Prelude
 - a. Prelude> :l Turing.hs
6. You should receive a message that the program has compiled and the prompt should have the word Main>

Enter main after the Main> prompt

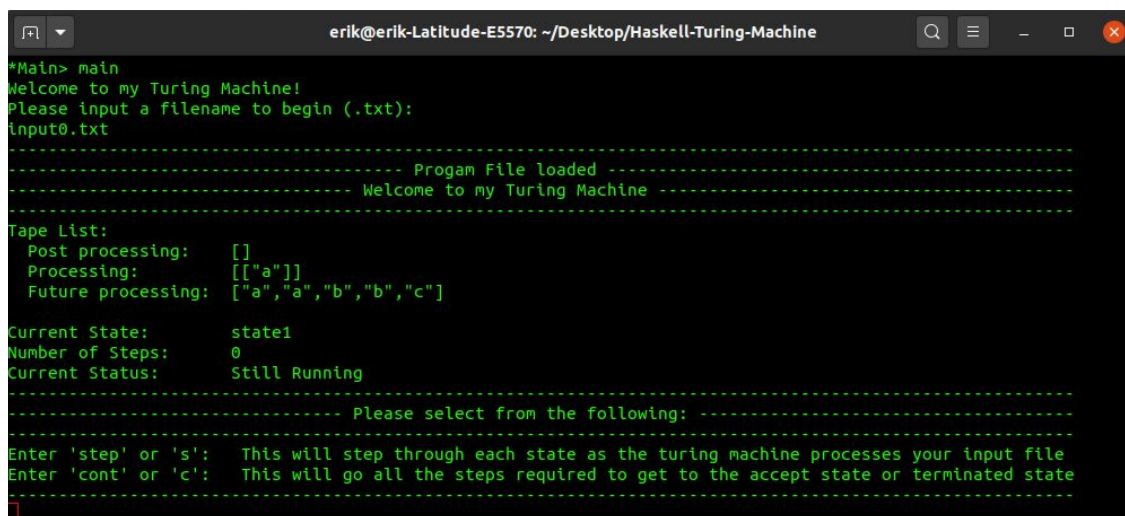
- a. Main> main



```
erik@erik-Latitude-E5570: ~/Desktop/Haskell-Turing-Machi...
erik@erik-Latitude-E5570:~/Desktop/Haskell-Turing-Machine$ ls
input0.txt input1.txt Turing.hs
erik@erik-Latitude-E5570:~/Desktop/Haskell-Turing-Machine$ ghci
GHCi, version 8.6.5: http://www.haskell.org/ghc/  :? for help
Prelude> :l Turing.hs
[1 of 1] Compiling Main             ( Turing.hs, interpreted )
Ok, one module loaded.
*Main> main
Welcome to my Turing Machine!
Please input a filename to begin (.txt):
```

7. This will then begin to run my Turing Machine Program, from there you can enter either of the input files that came from the zip file.

8. Assuming you enter input0.txt you should get the following:



```
*Main> main
Welcome to my Turing Machine!
Please input a filename to begin (.txt):
input0.txt
----- Program File loaded -----
----- Welcome to my Turing Machine -----
Tape List:
  Post processing:  []
  Processing:      [["a"]]
  Future processing: ["a","a","b","b","c"]
Current State:     state1
Number of Steps:   0
Current Status:    Still Running
----- Please select from the following: -----
Enter 'step' or 's': This will step through each state as the turing machine processes your input file
Enter 'cont' or 'c': This will go all the steps required to get to the accept state or terminated state
```

9. Lastly we can step/continue through the input file until we reach either an accept state or a trap state. From there the program is locked into that state until the user starts the program over again.

3. Research findings:

For this project I had to do a decent amount of research, mainly on haskell and how to implement a Turing machine in this language. I have made finite state machines and non-deterministic finite state machines in java. However I had a much harder time implementing a Turing machine in haskell. This was mainly because of my lack of familiarity with haskell. Most of my understanding for haskell came from the website learnyouahaskell.com. This website allowed me to go back and revisit code that was crucial for my project. From there I spent a good amount of time on stackoverflow.com, which led to other websites and was a great resource to layout how i could implement my project. There were various great one on one explanations, that provided code which really helped me understand how others were able to accomplish a similar task to the one I was performing. From there I was able to start building my machine. Like I said earlier I attempted to have the user enter all the information at the command prompt, however after many failed attempts I realized that almost all of what i found online was using an input file to receive user input. This led me in the right direction to where I was able to get a working program from my project.

4. Difficult parts of the project:

As stated earlier my biggest hurdle was how to receive, and process user data. Once I found out that I would be using an input file to receive user data, then it led to another hard task in my project; how to process the input file. Next I had to figure out how to parse through the file to get the appropriate information for the program. I have numerous functions that help me parse through the file to revoke values after the head has processed them. From there I had to determine where the start state was, the accept state, and any trap states. Next I had to build the tape with the values from the file and confirm those values with the given alphabet. After I got that working I tackled my next hurdle which was properly implementing the transitions. This proved to be a harder task than Initially expected. Luckily I was about to find a decent amount of literature on how to make the transition work for my program. Lastly I worked on getting a good looking user interface, and testing for bugs.

5. Future extensions:

I would say the future intentions for my project would be to go back and see if i could get the program to work without an input file, as i had originally planned. I'm sure it can be done, and now that i have gotten it to work with an input file, I know exactly what I need the user to enter from the command line prompt. Also I would test my code more to see if any more bugs arise from special case user data.

6. Conclusion:

In conclusion I have learned a decent amount about haskell and how to implement real world applications with it. This project has made me learn a new language that I'm definitely not familiar with, and prove that I can make a working product from it. It was also nice to revisit turing machines again and learn the importance of this topic. I'm excited that I was able to create a useful haskell program and hope to continue learning new and exciting languages as I continue my career in computer science.

Resources

Running Turing Machines in Haskell, 10 Jan. 2015, gedenkt.at/blog/turinghaskell/.

Learn You a Haskell for Great Good!, learnyouahaskell.com/.

Njlochner. "Njlochner/haskell_turing_machine." *GitHub*, github.com/njlochner/haskell_turing_machine/blob/master/Turing.hs.

Akroy. "Akroy/Haskell-Turing-Complete." *GitHub*, github.com/akroy/haskell-turing-complete/blob/master/TuringMachine.hs.

"Generate All Possible Turing Functions of a Type in Haskell." *Stack Overflow*, 1 Dec. 1968, stackoverflow.com/questions/58422064/generate-all-possible-turing-functions-of-a-type-in-haskell.