Ryan Darras PJ05 project writeup

Preface

This project was developed using Visual Studio 2017 community edition in C#. There are no customizations made that would interfere with grading or someone else opening and running the project. By default, when the program is run it saves the files into this location ~\bin\Debug\Outputs however I have copied this output into the results folder at the top of the zip file. Similarly, that same location except instead of \Outputs, do \Machines in order to read in the machine files. It is currently hardcoded to read in m00.tm, m01.tm, etc so if you don't have all 4 in the Machines folder it won't work. The projects starting file is program.cs. I have commented most of the code in the solution to where it shouldn't be too difficult to understand the strategy as you read along.

Abstract

All in all, this project was much easier than PJ01 in my opinion. Probably because a majority of the thought processes that went on in PJ01 was the same/similar to this one, so I already had it figured out. The overall goal of this project is as follows:

1) Load and save the input file as a list of strings. Call this "INPUT".
2) Create a new TM and send it a list of strings "MACHINE" that equates to the machine files. Example item in this list would be "0,98,1,97,R".
3) TM constructor reads each line in the list of strings in MACHINE and deconstructs them into FromState, ReadSymbol, ToState, WriteSymbol, and HeadDirection.
4) Uses the above variables to create a Transition struct and stores all of these objects in a Dictionary<int, List<Transition>> which is a dictionary of transitions, and the key is the from state. IE: When we are in state X, we can look up the transitions that are possible from said state.
5) After the TM is built, we run each string in INPUT over the machine and store the results in a List<string> output. (More info in next section)
6) Print output to the output files line by line.

Running the TM

To run the machine, you give it a single string as an input and it will do these steps:

1) Copy the input string onto the tape.
2) Loop until we get to a halting point (accept/reject/no transition)
    a) Figure out what transition we need to make based on current state and tape head.
    b) If we have a valid transition, continue; otherwise reject.
    c) Update our CurrentState to the next state.
    d) Write our new symbol to the tape at tape head location
    e) Update tape head location based on the transition data.
        i) If tape head location == 0, it stays at 0.
        ii) If left, it goes left.
        iii) If right, it goes right.
    f) If we are transitioning right, we might need to add a blank space to our tape because I chose to use a list as the tape. Handle that here.
    g) Check to see if we are in the accept or reject states. If so, halt.
3) Remove any blanks from the end of the tape and return it as the output.

Conclusion

Well… I know that this writeup is supposed to be ~2-3 pages, but there really isn't much more to say. This project felt like a spinoff of the first and thus it was fairly simple.