Multitape Deterministic Turing Machine

181240020 Hu Junhao

1 Project Progress

Three tasks has been completed and tested in great detail.

2 Analysis and Design for MDTM

The first step is to parse arguments passed from the command line. We made a good use of getopt_long functions in C.

The next step is to analyze the given tm file. We have compared different aspects of using C++ string and using C char arrays, and finally decided to stick to the char arrays, which might potentially bring us more convenience and overall control during coding. Specifically, it is much easier to trace the index of each tape head.

Then, using fscanf, we store every piece of information in char arrays, including input symbol set, tape symbol set, state, accepting state and so on.

So far, the parsing part is completed. Next, we enter the simulation part.

We also used char array as tapes and keep track of heads for different tapes. In addition, we keep track of the current state, the leftmost and rightmost position each head has traveled and so on.

For starter, we copy the input string on tape 0, and constantly executing the following loop until we stop at a halting state. For each loop, we look up the transition table, make comparisons between the current state and tape symbol with those in the transition table, and decide where to go next.

Finally, some error checkings and verbose printings are inserted among the original code. For verbose printings, we used an integer array to store the distance information. Using this information, we can make sure that all the index and symbols outputed are properly aligned.

3 Analysis and Design for Two Turing Programs

3.1 L1 — case 1

First, we need to copy the fisrt part of $a^i b^j$ to tape 1 and at the same time delete them from tape 0. In this process, we need to make sure that the strings copied

from tape 0 to tape 1 is indeed in the shape of $a^i b^j$. If there is an exception, we jump to the reject state.

Then, we can make comparison between the strings on tape 0 and tape 1. If there is a mismatch, we also jump to the reject state.

Only when we are sure that the strings on two tapes are exactly the same, we enter the accept state.

3.2 L2 — case 2

Fisrt, we need to copy the fisrt part of '1's to the left of 'x' to tape 1 and at the same time delete them from tape 0. In this process, we need to make sure that the strings copied from tape 0 to tape 1 is indeed '1's. If there is an exception, we jump to the reject state.

Then, we start to delete '1's to the right of 'x'. Whenever we delete one '1', we copy the '1's on tape 1 to tape 2 once. During the process, if we meet any exceptions before encouter '=', we jump to reject state.

When we finnaly encounter '=' on tape 0, we start to make comparison between strings on tape 0 and tape 2. If they are the same, enter accept, or enter reject.

4 Conclusions and Suggestions

This project allows me to fully understand what turing machine is and how it works, and I really appreciated it. However, as an open project, we hope to at least have a general picture of the input sizes, or else with literally no assumptions, the real work is hard to even get started. For example, in my work, the tape is not infinite (of course since it is implemented on a single computer), the maximum length of which is just around 100000 characters (can be altered as a super parameter).

Although we can come up with all sorts of ways to tackle with this "infinity problem", like dynamically allocating memory blocks, the overwhelming complexity will somehow dilute the subject of this project.