Sophie Lama
8 December 2024
Theory of Computing
Project 02: Tracing NTM Behavior

Detailed Discussion of Project 02 Results

How your code works:

Running the main program begins with asking the user for some input. The user can enter the name of the CSV file of the Turing machine they want to run, as well as an input string to run on the machine. They also enter in the limit, which is the maximum depth for the tree of possible configurations.

Next, the CSV file is read, and the attributes of the machine are added to a Machine object and returned to the main function. After printing some initial information about the machine and string, we begin to iterate through the tree of possible configurations, trying to find a path that leads to an accept state.

This iterate function is recursive, and it works by first returning if the current depth has exceeded the limit. This allows the program to break out if the maximum depth has been reached. Then, the function loops through the Turing machine's list of possible transitions. It searches for any transitions that match up with this current step's state and the current character pointed to by the head pointer. It creates a list of matches that will be used later.

If, after the looping, there was no match found for the current state and character, then we pop this current configuration from the list of configurations, since this is a dead end of sorts in the tree. We then update the global variable total_transitions since we are transitioning into a reject state, then return from the function.

If we did find at least one match, we update the depth of the tree, since this means we'll be going down a level and making at least one transition. We then do a quick loop through the list of matches to see if any leads to the accept state. If there is a match that leads to acceptance, then update the total transitions, update the list of configurations, and return from the function with the accept state, depth, and list of configurations.

If no possible match leads to an accept state, then we simulate going down each possible path, calling the iterate function within itself to do so. If any of these paths leads to acceptance, then return that path. Otherwise, update the stat_list global variable and add each return to the list of statuses. Once we are done going down each possible path, we iterate through stat_list to find the rejection with the largest depth, then return that one.

The iterate function returns the final state of the best possible path, as well as the path's depth in the tree of configurations and the sequence of configurations for the path. If this was an accept path, then the list of configurations will be printed, as well as the depth of the accept path. If this was a path that exceeded the depth limit, this information will be printed to the user. Finally, if no accept path was found and none exceeded the limit, then the longest reject path's depth will be printed out to the user.

How the code models nondeterminism (how it measured nondeterminism in each run):

This program models nondeterminism in the iterate function. This is a recursive function that calls on itself internally. It simulates going down every possible path in the

tree of configurations that the input string could have travelled. This models nondeterminism by simulating the possible "choices" that a Nondeterministic Turing Machine would have to make. This program doesn't have a crystal ball to predict which path is the best possible path, so it instead follows every possible path.

How you determined your program was correct (how you "traced" your machine's execution and printed out intermediate steps):

I tested this program by running it with simple input strings and including *many* print statements within my functions. I was mostly keeping track of branch depths, the head pointer, and the list of configurations, making sure those were all accurate by comparing them to what I had calculated by hand that the TM should be doing.

What test cases you used (how they demonstrate correct behavior):

I used two test files provided by other students, one for the language of strings of the form a*b*c*, and one for the language of strings of the form a^+. I used those CSV files with my program and drew on paper what their state machines might look like. As I was writing my code, I was comparing it to my state machine to see if it was iterating properly through the Turing machines.

How to read any output traces that showed execution:

I have two text files with output from the program, one involves the a*b*c* test Turing Machine, and the other involves the a^+ machine. For each instance the program was run on an input string, there are lines of text that detail the name of the machine, the string inputted by the user, as well as the results of the tracing. Those results include the depth of the tree, the total number of simulations run, and whether the string was ultimately accepted or rejected. If the string was accepted, the list of configurations the string went through is also included.