UNIVERSAL TURING MACHINE

COMSC 330 FALL 2019

Abstract

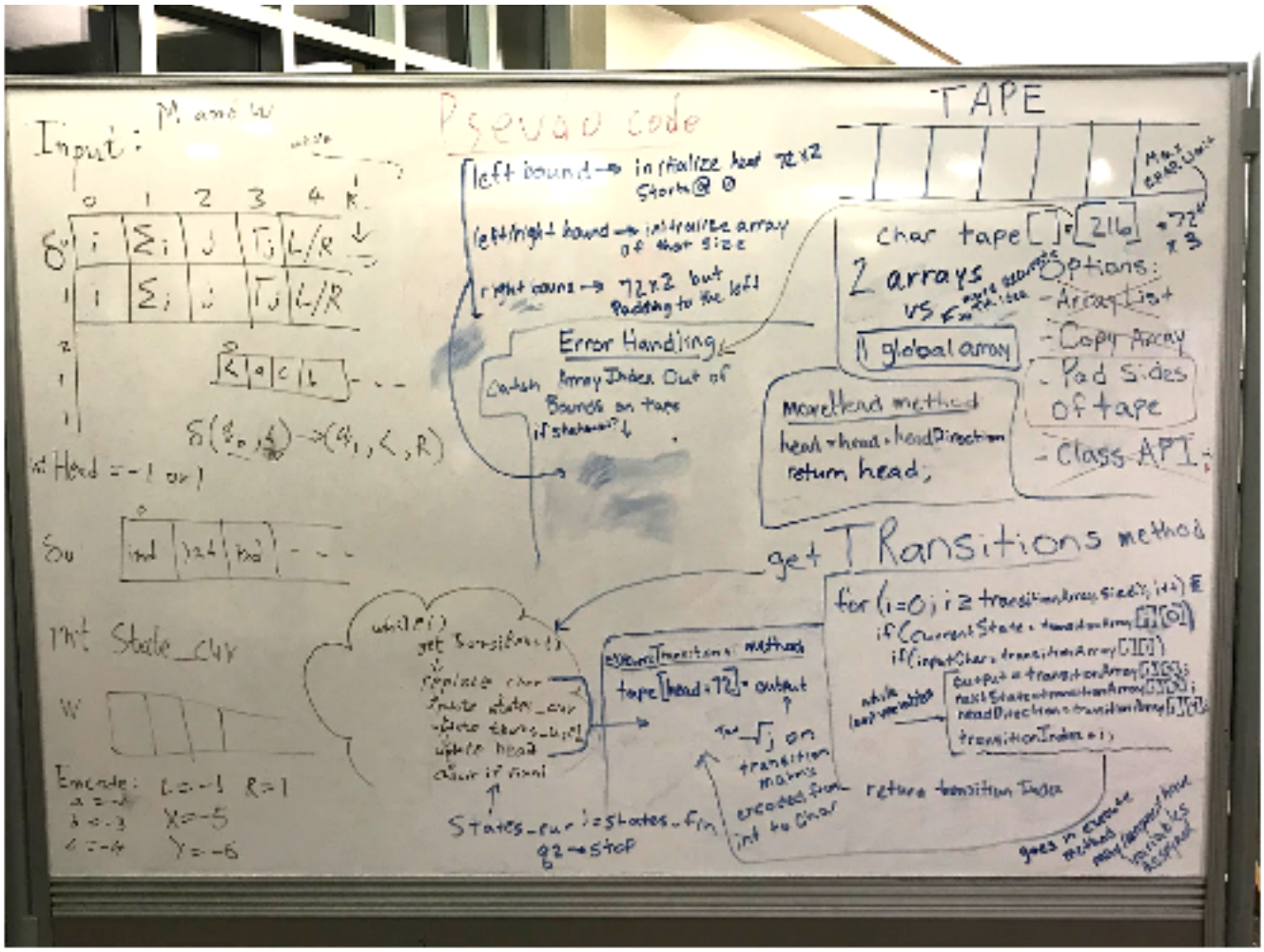
Jordan McGarty, Kristina Mendela, Dhaval Patel

This project was done to simulate the usage of a Universal Turing Machine (UTM). This is used to simulate any kind of arbitrary turing machine on an arbitrary input. We were assigned to recreate this process.

Executive Summary

This project was assigned as part of the COMSC 330 course Software Design. The purpose of the project is to use previous knowledge of Universal Turing Machines from Theory of Computation and implement it in a software development team. The methods and process of designing a software in a group setting simulates the environment of software engineering done for complex real world problems. The team effort for the comprehensive final product emphasizes the documentation and organization of teams in a typical software development environment. Implementation of Universal Turing Machines serves as a good platform to apply Theory of Computation and software design principles for preparation of the professional setting.

Currently the team has wrapped up developing the Universal Turing Machine using the previously planned pseudo code; as a whole we are in the final stage of testing. We are double checking our code, searching for errors, or any missed requirements previously overlooked. To recap on what we have done to reach this state we have met as a group to further develop our planning as shown in the powerpoint demonstrations, which I will recopy here:



This is the full plan for our Universal Turing Machine excluding some details inside the “puffy cloud.”

Contents

Abbreviations and Glossary

UTM - Universal Turing Machine

Introduction

This report is showcasing the primary aspects that went into developing a simulation of the Universal Turing Machine. Within this report you will find all the necessary information in regards to our processes, issues we ran into, requirements, etc. This project was done using prior knowledge to strengthen our understanding of the Universal Turing Machine, in addition to software design as a whole.

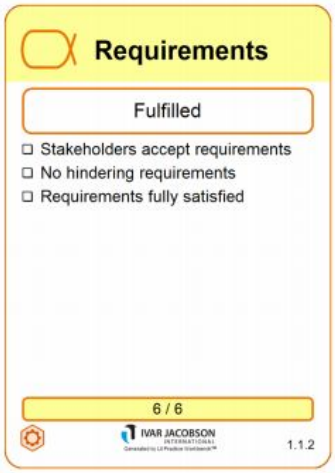
This report will also be frequently updated based on the progress we make to keep the information fresh and up to date. We will also be improving sections to further showcase our understanding of the material.

Requirements

The primary requirements for the Universal Turing Machine were as follows:

- Develop a program that simulates a Universal Turing Machine
- Allow the user to enter their own Turing Machine and data stream
- Program must encode each transition of the user's TM and describe how it would establish the Turing Number of the user's TM
- The program must somehow demonstrate the step-by-step execution of the user's TM based on the input data
- It must determine if the data is accepted or not
- Indicate the machine's state and the final tape

In addition to this, we are making progress through essence; here we are making progress with our requirements. We have fulfilled the requirements for the UTM. There are no hindering requirements and they are fully satisfied.

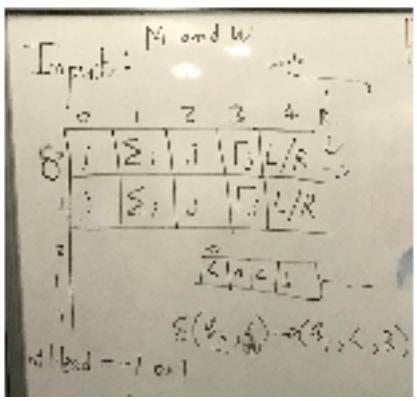


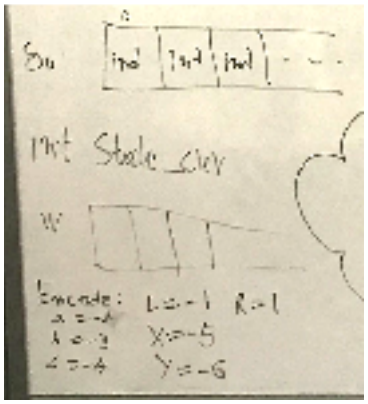
Design & Implementation

To begin designing the UTM, we started by breaking it down into components. The components include parts such as the tape, states, alphabet, transitions, and language. After we broke the UTM down, we started to discuss the architecture for each separate component. For example, we talked about the tape, and how that could be implemented by taking an input string from the user and then storing that string in a character array where each character has its own cell. From here we discussed our usage of the essence opportunity. Our opportunity has been fully addressed, our solution is worth deploying, and the stakeholders are satisfied.

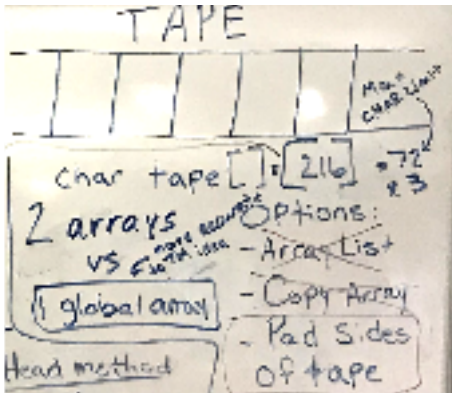
After several weeks of developing we have decided on how we will structure the Universal Turing Machine. We have posted a picture of our “blueprint” for the universal turing machine above in the executive summary, but here we will break it down.

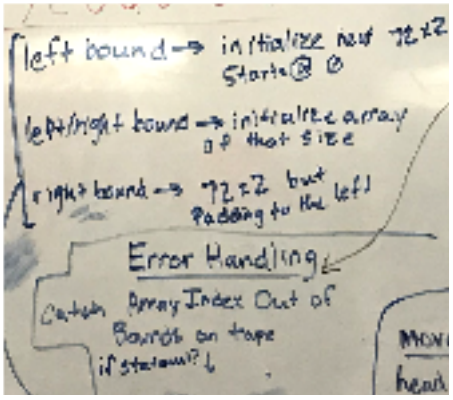
For the input, we will ask the user which states are final and if the tape is bounded on either side. We will also get the input string and description of the machine from the user. The input is the description of the machine, M, and the input string, W. The transitions are going to be input into a 2D character array, where it will read like a transition diagram, where i = currentState, Σ_i = inputChar, j = nextState, Γ_i = outputChar, and L/R = headDirection. We are going to keep track of our used transitions by storing the index of every transition we match in its own array as a record of step-by-step execution.



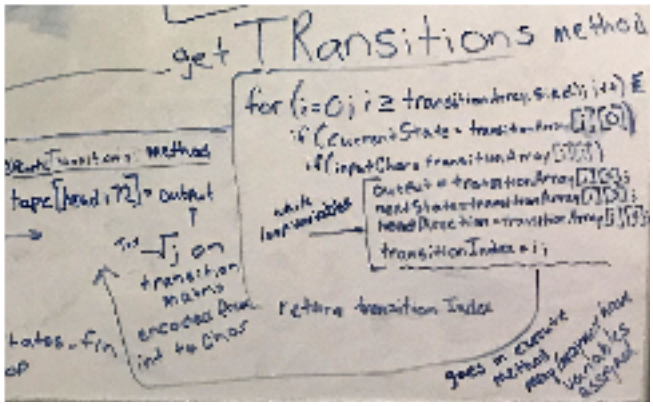


For the tape, we decided on a single global array as opposed to our original plan of 2 separate arrays for input and output. We decided this because it would be more accurate to a Turing Machine to only use a single tape. In order to buffer the head so that it won't easily hit an Array Index Out of Bounds exception, we padded the sides of the tape. We did this by letting 72 chars be the tape size for the input, and then allowing for another 72 chars on either side so the head can move. Even though we added padding to avoid Array Index Out of Bound errors, we also implemented exception handling where the code is most likely to throw that error. When the head is off the scope of the UTM and attempts to read in an input character from the tape, the program will throw an Array Index Out of Bounds exception. However, our try-catch statement will catch this error when it happens and will copy the tape onto a larger tape and then retry the transition. This not only helps minimize errors, but also better simulates an infinite tape when unbounded.





We are handling tape bounds during the tape initialization process, which has its own module. The tape is initialized differently based on whether the Turing Machine is left bound, right bounded, both, or unbounded. We ask the user during the input process for the bounds, and the bounds string is passed to the initializeTape module. Here, we go through a series of if-else statements searching for a match to the bounds string. If the TM is left bound, we initialize the tape at a size of [72 + 72], and input the char ">" directly on the tape at [0], unless the bound character is already within the string. The head is then initialized at the index [1]. If the TM is right bounded, we initialize the tape at size of [72 + 72], and input the "<" directly on the tape at [tape.length - 1], unless the bound character is already within the string. The head would then be initialized at the length of w away from the right end of the tape (the head starts at the first char on the tape). If the TM is linear bounded, we initialize a tape the size of [w.length + 2] . We then input the ">" at tape[0], and "<" at tape[w.length-1] unless the string already contains bound characters. If the TM is unbounded, we initialize a tape of size [72 + 72 + 72]. We then initialize the head at index [72], or the middle of the tape.



For transitions, after the Turing Machine has been defined, and we received our input char from the user, we have to match the currentState and input char to one of the transitions. Our transitions are implemented in a 2D array, so we created a module to extract the correct transition from the 2D array and a module for executing the transitions. *getTransitions* makes the comparisons to choose the correct transition, then stores the index of that transition in a usedTransition array. This index is then passed to the *executeTransition* method which includes changing the current state, writing to the tape, and moving the head.

Development Environment

Our development environments are as followed:

HP Notebook running Windows 10, with an AMD A-12 processor with 12GB of RAM and using Java programming language.

HP Spectre x360 running Windows 10, with an Intel i7 processor and 16GB of RAM and using Java programming language.

Acer Aspire e15 running windows 10, using an Intel i5 processor with 8GB of RAM and using Java programming language.

Target Environment

The target environment is the same as the development environment due to the lack of a formal deployment plan.

Testing

Our plan to test our Universal Turing Machine is to run the program on all three devices separately to determine that the program is successful in multiple environments. In addition to this we plan on testing multiple strings to confirm that the outcomes are indeed correct. We have multiple test cases and transitions prepared to be used in confirming the Universal Turing Machine is running correctly. In addition to this we have transition diagrams for test cases to make sure everything is running accordingly. We plan on making minor adjustments to test each scenario. In addition to our previous adder test case, we are going to run tests using the example acceptor Turing Machine that was sent by our stakeholder and client, Dr. Ruocco. In these tests, our language is $a^n b^n$.

The test Turing Machine is as follows:

```
1,a,3,_,R
1,b,7,0,R
1,_,2,1,R
3,a,3,a,R
3,b,3,b,R
3,_,4,_,L
4,a,5,_,L
4,b,6,_,L
4,_,2,0,L
5,a,5,_,L
5,b,5,_,L
5,_,2,0,L
6,a,6,a,L
6,b,6,b,L
6,_,1,_,R
7,a,7,_,R
7,b,7,_,R
7,_,2,_,R
w = aabb
Final State: 2
```

The Universal Turing Machine will take this input and execute it to get an end state of 2.

```
The Turing Machine starts at state 1
The Turing Machine is entering state 3
The Turing Machine is entering state 3
The Turing Machine is entering state 3
The Turing Machine is entering state 3
The Turing Machine is entering state 4
The Turing Machine is entering state 6
The Turing Machine is entering state 6
The Turing Machine is entering state 6
The Turing Machine is entering state 1
The Turing Machine is entering state 3
The Turing Machine is entering state 3
The Turing Machine is entering state 4
The Turing Machine is entering state 6
The Turing Machine is entering state 1
The Turing Machine is entering state 2
HALT


Output Tape: [1,]

The transitions used were: [1,4,5,5,6,8,14,13,15,1,5,6,8,15,3,]
```

An output of [1] means that the string is accepted by the Turing Machine.

We are now going to test our UTM with a string that should not be accepted. With the same transitions and final state, we are going to run the machine with an input string of w = ababa. The results are as follows:

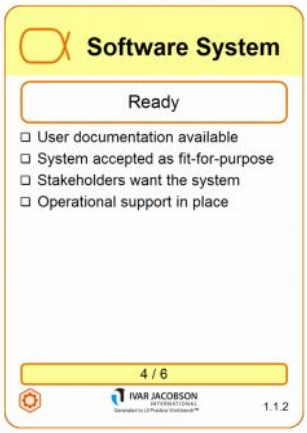
```
The Turing Machine starts at state 1
The Turing Machine is entering state 3
The Turing Machine is entering state 3
The Turing Machine is entering state 3
The Turing Machine is entering state 3
The Turing Machine is entering state 3
The Turing Machine is entering state 4
The Turing Machine is entering state 5
The Turing Machine is entering state 5
The Turing Machine is entering state 5
The Turing Machine is entering state 5
The Turing Machine is entering state 2
HALT


Output Tape: [0,]

The transitions used were: [1,5,4,5,4,6,7,11,10,11,12,]
```

As we can see the output tape contains only a 0, which means that the string is not accepted by the defined Turing Machine. After testing a few acceptable and non-acceptable strings, we can determine that our UTM passes our tests using the example acceptor Turing Machine.

In our essence cards, our software system is ready. The system is fit for purpose and the stakeholders want the system.



Deployment

There is no formal deployment plan, however this project could potentially be deployed to the students in COMSC 355 Theory of Computation, so the program will have to be prepared for deployment.

Summation and Conclusions

At the beginning of the project, we all had different concerns that we ran into. Dhaval is currently taking Theory so he had to catch up on the knowledge Kristina and Jordan had prior. Jordan had to work on improving his syntax for the coding aspects, and Kristina had to work on improving her theoretical knowledge. As a whole we all were in need of additional research to fully understand the UTM.

Now at the end of the project, we have all gotten caught up on the concept of the UTM and are all on the same page with how the code is formatted. Getting together as a group has helped to enhance the understanding and unity of our thought processes. We are done with the design of the project and have finished the implementation and debugging. The next steps are to deploy the UTM.

In regards to non determinism within a turing machine; with multiple options or branches to different states, the UTM will replicate the transitions until they run to a final state. Essentially, it will simulate an exponential growth when it runs. This is because it can branch off to multiple different paths and will replicate the transitions multiple times with different turing machines. The implementation while possible, is not feasible due to most computers not being able to handle such a large amount of turing machines running at one time. In the event of an expanding alphabet, the change should not affect the Universal Turing Machine if the transitions are correctly matched to the alphabet. Additionally, the expanding Q (or states) will not affect the Universal Turing Machine as long as the transitions are correct.

Acknowledgements

Professor Ruocco

References

"Building a Universal Turing Machine." *YouTube*, Udacity, 23 Feb. 2015, www.youtube.com/watch?v=C9T6iQYCW10.

Theory of Computation Powerpoints and resources

De Mol, Liesbeth. "Turing Machines." *Stanford Encyclopedia of Philosophy*, Stanford University, 24 Sept. 2018, plato.stanford.edu/entries/turing-machine/.

Jacobson, Ivar, et al. *The Essentials of Modern Software Engineering: Free the Practices from the Method Prisons!* Association for Computing Machinery, 2019.

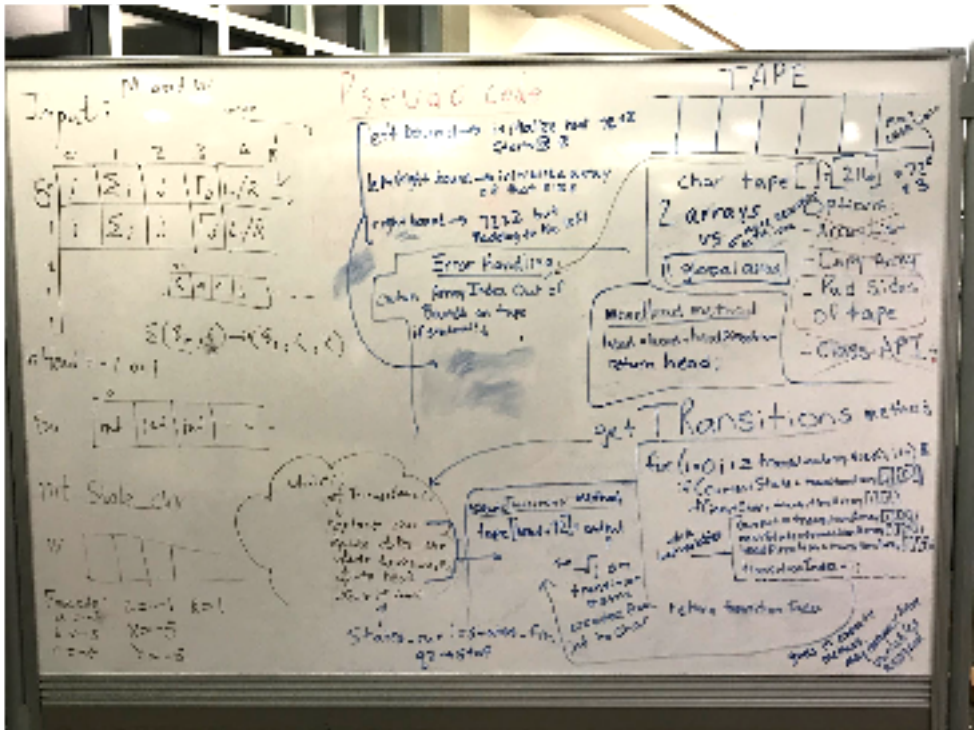
Sommerville, Ian. *Engineering Software Products*. Pearson, 2020.

"Universal Turing Machine." *Universal Turing Machine - Rosetta Code*, rosettacode.org/wiki/Universal_Turing_machine.

Appendices

Requirements

Design



Tests

We conducted a test using an unbounded adder Turing Machine. Our unbounded Turing Machine implementation passes the test. Here are the transitions, the input string, and the final state for the Turing Machine:

```
0,a,3,_,R

0,b,1,_,R

1,a,1,a,R

1,b,1,b,r

1,_,2,b,l

2,a,2,a,L

2,b,2,b,L

2,_,0,_,R

w = bbabbb

final_state = 3
```

The results of the test are as follows:

```
The Turing Machine starts at state 0
The Turing Machine is entering state 1
The Turing Machine is entering state 1
The Turing Machine is entering state 1
The Turing Machine is entering state 1
The Turing Machine is entering state 1
The Turing Machine is entering state 1
The Turing Machine is entering state 2
The Turing Machine is entering state 2
The Turing Machine is entering state 2
The Turing Machine is entering state 2
The Turing Machine is entering state 2
The Turing Machine is entering state 2
The Turing Machine is entering state 0
The Turing Machine is entering state 1
The Turing Machine is entering state 1
The Turing Machine is entering state 1
The Turing Machine is entering state 1
The Turing Machine is entering state 1
The Turing Machine is entering state 2
The Turing Machine is entering state 2
The Turing Machine is entering state 2
The Turing Machine is entering state 2
The Turing Machine is entering state 2
The Turing Machine is entering state 2
The Turing Machine is entering state 0
The Turing Machine is entering state 3      _, b, b, b, b, b, _,

The transitions used were: [1, 3, 2, 3, 3, 3, 4, 6, 6, 6, 5, 6, 7, 1, 2, 3, 3, 3, 3, 4, 6, 6, 6, 6, 5, 7,
```

We conducted another test using an acceptor Turing Machine provided by Dr. Ruocco. The description of the machine is as follows:

```
1,a,3,_,R

1,b,7,0,R

1,_,2,1,R

3,a,3,a,R

3,b,3,b,R

3,_,4,_,L

4,a,5,_,L

4,b,6,_,L

4,_,2,0,L

5,a,5,_,L

5,b,5,_,L

5,_,2,0,L

6,a,6,a,L
```

```
6,b,6,b,L

6,_,1,_,R

7,a,7,_,R

7,b,7,_,R

7,_,2,_,R

w = aabb

Final state: 2
```

The results of the test are as follows:

```
The Turing Machine starts at state 1
The Turing Machine is entering state 3
The Turing Machine is entering state 3
The Turing Machine is entering state 3
The Turing Machine is entering state 3
The Turing Machine is entering state 4
The Turing Machine is entering state 6
The Turing Machine is entering state 6
The Turing Machine is entering state 6
The Turing Machine is entering state 1
The Turing Machine is entering state 3
The Turing Machine is entering state 3
The Turing Machine is entering state 4
The Turing Machine is entering state 6
The Turing Machine is entering state 1
The Turing Machine is entering state 2
HALT


Output Tape: [1,]

The transitions used were: [1,4,5,5,6,8,14,13,15,1,5,6,8,15,3,]
```

We also tested a string that would not be accepted by the Turing Machine. We chose w = ababa. The results of the test are as follows:

```
The Turing Machine starts at state 1
The Turing Machine is entering state 3
The Turing Machine is entering state 3
The Turing Machine is entering state 3
The Turing Machine is entering state 3
The Turing Machine is entering state 3
The Turing Machine is entering state 4
The Turing Machine is entering state 5
The Turing Machine is entering state 5
The Turing Machine is entering state 5
The Turing Machine is entering state 5
The Turing Machine is entering state 2
HALT


Output Tape: [0,]

The transitions used were: [1,5,4,5,4,6,7,11,10,11,12,]
```

An output of [0] means that the string was not accepted by the Turing Machine.

Planned and Actual Schedule

The schedule we followed during this project was meeting on Wednesdays and Thursdays in the afternoon or evening. We also worked online individually on shared documents to maintain a steady progress. In addition to this we planned on meeting more frequently when needed if falling behind occurs, or if conflicts naturally happen. Fortunately, we did not have many instances where we felt like we were falling behind and there was minimal conflicts.

We also have been meeting at the end of class making usage of the time given to us. With this time we do smaller meetings just for minor things to go over as well as help plan other meetings.

Manuals

Entering the Input

When using the Universal Turing Machine the user must give the description of the machine. To start:

1. The user must enter the amount of transitions. Transitions are entered in the format (i, Σ, j, Γ, L or R) in a list with 1 transition per line. Instead of spaces, we use underscores (_). For example, 4,b,6,_,L, where 4 is the currentState, b is the input character, 6 is the state we are moving to, _ is the output character, and L is the direction of the head movement. When the user is finished entering transitions, the user can press 'T' or the UTM will naturally move on once you've entered the specified number of transitions.
2. Then the user must specify what the final state is.
3. Then the user must enter the input tape characters without spaces or commas.

4. Lastly, the user must specify if the tape is left-bounded, right-bounded, linear bounded, or unbounded by inputting characters such as “l” for left-bound, “r” for right-bound, “lr” for linear bound, or “u” for unbound. Please be sure to input the correct bound for your Turing Machine, as the bound could affect the rest of the execution.

Interpreting the Results

After the Universal Turing Machine computes, the answer you see will look like the following:

```
The Turing Machine starts at state 1
The Turing Machine is entering state 3
The Turing Machine is entering state 3
The Turing Machine is entering state 3
The Turing Machine is entering state 3
The Turing Machine is entering state 4
The Turing Machine is entering state 6
The Turing Machine is entering state 6
The Turing Machine is entering state 6
The Turing Machine is entering state 1
The Turing Machine is entering state 3
The Turing Machine is entering state 3
The Turing Machine is entering state 4
The Turing Machine is entering state 6
The Turing Machine is entering state 1
The Turing Machine is entering state 2
HALT


Output Tape: [1,]

The transitions used were: [1,4,5,5,6,8,14,13,15,1,5,6,8,15,3,]
```

The Turing Machine begins in state 1, and then notifies the user whenever it moves states and when it halts.

The ‘Output Tape’ shows what the tape looks like at the end of the execution. What is written on the tape at the end of the execution determines whether the input was accepted.

The ‘transitions used’ keep track of exactly which transitions are used while the Turing Machine is running. This can be used as a record of the step-by-step execution of the UTM.