SDP - Turing Machine Project

June 2024

1 Description

A web app that can be used to learn and understand Turing Machines as state machines. The tool will help students better learn the capabilities of Turing Machines, and to help automate and improve the marking process for Turing Machine questions in the COMS3003A Formal Languages and Automata module.

The tool must provide visual TM programming, a scriptable language, tests and some debugging tools. Additionally, it can provide some useful information and usage visualization of a TM's behaviour.

1.1 TM Conventions

1. Single sided tape
2. User specified tape alphabet, but will always have $\sqcup$ and $\triangleright$
3. User specified input alphabet of size at least 1

2 Requirements

2.1 Visual Turing Machines

1. A visual graphic of the TM's state machine (see figure 1)
2. A visual graphic of the TM's tape (see figure 2)
3. A timestep indicator (number of executed steps)
4. A used tapecell indicator (number of used tapecells)
5. TM Construction
    - Add / Remove States (Drag-and-Drop)
    - Add / Remove / Edit transitions (Drag from qi to qj)
    - Add / Remove Tape Alphabet Symbol (button on side)
    - Add / Remove Language Alphabet Symbol (button on side)

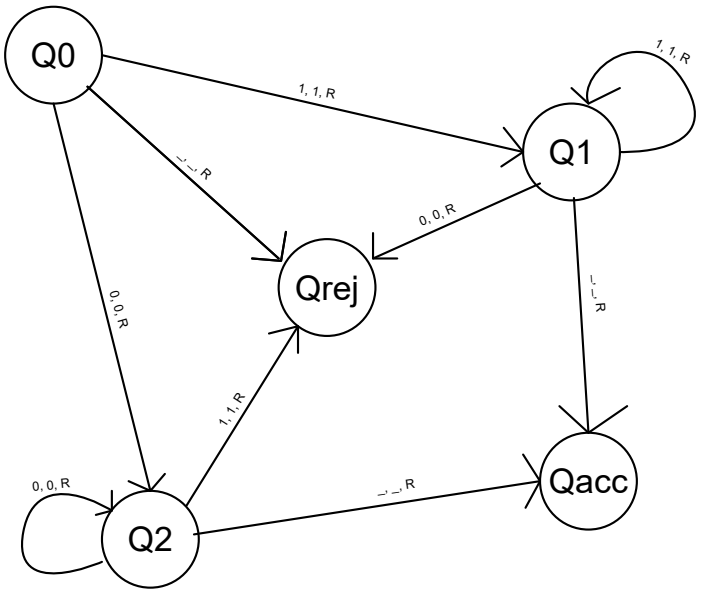

Figure 1: Example Turing Machine Visualization

6. TM Execution
    - Visual execution of the TM with a highlighted state and animation of moving tape
    - Execution speed parameter (modify how quickly it moves through one step of the execution)
    - Button for instant execution (as fast as possible with no animation / visual component)
7. Download TM - Download a TMS file of the current TM (see below for TMS)
8. Upload TM - Upload a TMS that then has the TM drawn in graphically
9. Copy Program Button - Copy the binary encoding of the TM's instruction set

2.2 TM Script

A scripting / programming language that encodes a Turing Machine's execution. It is a more expressive form than just using the TMs instruction set encoding.

This language will be defined more concretely at a later date. For now, the language will have syntax (and a potential example) for the following:

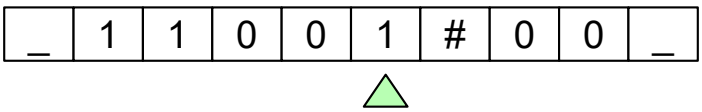

Figure 2: Example Tape Visualization

1. Symbol sets - Tape and Input symbols (‘input_sym = [a, b, ...]’)
2. State set (‘states = [q0, qAccept, qReject, ...]’)
3. Transitions (‘qi,sk,qj,sl,D’ or ‘qi,sk $->$ qj,sl,D’)
4. Test cases (‘test { input_word,expected_status, max_time,max_space }’)
5. Metadata (‘meta { title, language_description }’)
6. Comments (// or #)

This language is not really intended for direct coding. The language is predominantly for a download or upload of a TM through the visual editor. However, rather than using an encoding of the TM’s instruction set, this language can provide some additional information of the Turing Machine.

The language should be expressive enough to capture potential mistakes from a user. For example, ignoring a specific tape symbol in the instruction set should not remove that symbol from the Tape Symbol set because it is a mistake that a student may be penalised for in a test / assignment. Therefore, the tape symbol set must be specified in the file.

2.3 Test

1. Test Case:
   - Input word
   - Expected result (Accept / Reject / Timeout) *Timeout is specified by a TM that reaches max timesteps / tapecells*
   - Optional: Max number of timesteps in computation
   - Optional: Max number of tapecells to be used in computation
2. Test Suite:
   - List of test cases
   - Dependent on browser behaviour Parallel list of weights
   - Dependent on browser behaviour Parallel list of seen / unseen test indicators
3. Test Suite specification in TMS
4. GUI:
   - Button to run current TM on test suite
   - Add / Remove / Edit test case
   - Button to save suite
   - Test case results and test suite results (summary of all tests)
5. Required browser behaviour: If test cases are stored client side, they must not be visible to the user via the browser console, inspect element or similar. i.e. students must not be able to view test cases aside from any GUI elements

2.4 Debugging

1. Add breakpoints in visual editor
    - Breakpoint can be placed on state (break as enter state)
    - Breakpoint can be placed on transition (break after transition is executed)
    - Breakpoint can be placed on tapecell (break as tapehead begins scanning cell)
2. Step Forward - Move forward one execution
3. Pause - break on the current state (without a breakpoint)
4. Continue - resume execution

2.5 Automarker

A program that can be used to read TMS and automark the results against a test suite. Must be compatible with Moodle.

Discussion with Pravesh required to better understand requirements for the automarker.

Note: Not necessarily for students as part of SDP, but does need to be completed.

2.6 Misc

1. Example TMs - A list of available TMs that demonstrate some technique
    - Note: If included, this should be something that can be disabled for an FLA test.
2. Universal Turing Machine
    - A manually coded universal TM that will demonstrate how the UTM can simulate another TM given as input.
3. Alternative Computing Models
    - Non-Deterministic TM's
    - Alternating TM's
    - Clock TM's
4. Modified instructions
    - Repeated direction transition D^n
    - Stationary transition S