Two-dimensional Turing Machine Experiment

The main goal of this experiment is to show and investigate how two-dimensional Turing Machine works and behaves for some given input, depending on how has the machine been programmed. This paper focuses on three different machines, analyses their specific behaviour and investigates couple of interesting executions. In this experiment, every machine's language is presented as a set of pictures, what makes it more user-friendly and easier to analyse.

Screen Size

Each of the machines included in the attachment uses grid of 5 columns by 4 rows as a two-dimensional tape to read input from and to write output to. Each “tile” or “block” of the tape consists of an image of size 117 x 117 pixels and an extra space around for border indicating in what state the machine is while reading particular input. Main window is resizable; however its preferred size is set to 800 x 600 pixels.

Implementation

All machines have been implemented in Java and do not require any additional libraries to run correctly (standard JDK is enough). Their alphabet includes 7 distinct images and one blank space. Each image has a number that it corresponds to. The mapping from numbers to images looks as follows:

<table>
<thead>
<tr>
<th>Number</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><img src="image0.png" alt="Image 0" /></td>
</tr>
<tr>
<td>1</td>
<td><img src="image1.png" alt="Image 1" /></td>
</tr>
<tr>
<td>2</td>
<td><img src="image2.png" alt="Image 2" /></td>
</tr>
<tr>
<td>3</td>
<td><img src="image3.png" alt="Image 3" /></td>
</tr>
<tr>
<td>4</td>
<td><img src="image4.png" alt="Image 4" /></td>
</tr>
<tr>
<td>5</td>
<td><img src="image5.png" alt="Image 5" /></td>
</tr>
<tr>
<td>6</td>
<td><img src="image6.png" alt="Image 6" /></td>
</tr>
<tr>
<td>7</td>
<td><img src="image7.png" alt="Image 7" /></td>
</tr>
</tbody>
</table>

Please note, those numbers have only been used to simplify implementation.

The program simulates three different Turing Machines and also allows running them simultaneously. First machine runs through the option menu called “Basic Algorithm” and is a simulation of the standard deterministic Turing Machine. It has two states (red and blue) and specified exactly one transition for every combination of state and input image. For most inputs
it changes the image and always moves to another “tile”. It halts only when it gets blank space as an input while being in the red state. The starting state for this one is a blue state.

Another machine implemented there is one called “Random choice”. The main idea of this one is to move in any random direction every time. The only specified behaviour is what image will be printed for a given input and to which state it will be changed to. It simulates non-deterministic Turing Machine (more than one transition for every combination of input and state). It also halts on a blank space while being in the red state (it has red and blue state only).

The last one, called “Dynamic Tiles” works on the same algorithm as the first one mentioned (“Basic Algorithm”), but one thing that makes it different, after every transition, one of the tiles from the tape (randomly chosen) is being changed to different one (also random). It provides some level of non-determinism, as it is not always certain in how many moves or after how much time the machine will halt.

The only difficulty encountered during implementation was how to make two or more machines (algorithms) running at the same time and how to run them to show the execution in “step-by-step” fashion. Fortunately, usage of threads in Java turned up to be simple and effective solution.

Moving off the screen problem

All machines have been implemented to handle moving outside the window bounds. When situation like this occurs, pointer to the “tile” is being wrapped around, i.e. if the machine wants to move up and is already at the top row, the next location that it will point to is the very bottom row of the same column (it works in the similar fashion when moving down out of bounds). Same thing happens when the machine wants to move left and is already at the very left column, it will be moved to the very right column of the same row (similarly for moving right out of bounds).

Experiments

- Basic Algorithm

1. For an input where at least one blank space occurs, in about 95% of cases the machine halts after some number of transitions. Images are being changed correctly, states are changing as well, everything works fine and the machine eventually gets to the blank input while being in a red state, what causes halt. It usually takes different number of transitions for each input. The least observed number of moves after which the program terminates in normal fashion is 1 move.
2. For the remaining 5% of cases, program catches itself in an infinite loop and moves only along one column or one row. It happens when column consists of only yellow and green images and the machine is in the red state while reading any of those inputs (figure 1.1) or when row consists of pink and blue images alternatively and the machine is in a red state when reading pink image as an input (figure 1.2). The sequence of images like this does not necessarily have to occur in the initial input, it can be created by the machine while executing the program. For this machine it is more likely that it will catch itself into vertical loop (moving along the column), than in the horizontal loop (moving along the row).

![Figure 1.1](machine loops on the very left column)

![Figure 1.2](machine loops on the second row from the top)
3. For an input, where no blank space occurs, we can certainly say, that the machine will never halt, as none of the transitions implemented for this machine produces output of a blank image. In other words, if the blank space doesn’t occur in the initial input, it is certain that it won’t get generated during the execution; therefore the machine will keep looping. The interesting fact is this, that during the execution, after some period of time, machine will catch itself in an infinite loop moving only along one column or one row (as in the examples above). For every test with completely filled up tape (no blanks), after some period of time the machine generated sequence of images that caused this vertical or horizontal looping, therefore it has been assumed that it will happen in 99% of cases (1% reserved just in case for an input that will not produce this kind of loop, however it has not been found during this experiment).

- Random Choice

4. For any input where at least one blank space occurs, the machine will terminate in normal fashion after some number of transitions. It is not certain and is not possible to predict after how many moves the machine will halt, as it has multiple choices where to go for every transition (this machine has been implemented as non-deterministic one). What is interesting, in 7 times out of 10, this machine requires more transitions to halt, than standard deterministic machine described above. The least observed number of moves after which the program terminates in normal fashion is 1 move.
   
   Also, for an input with no blank spaces, this machine will never catch itself in an infinite “vertical” or “horizontal” loop that has been mentioned earlier as a problem with deterministic “Basic Algorithm” machine. It has to do with non-determinism, as every time different choice where to move can be taken.

- Dynamic Tiles

5. For every input (even one with no blank spaces) this machine will always terminate in normal fashion after some number of transitions. After every transition, one of the tiles from the input (randomly chosen) is being changed to different one (also random). The machine has been implemented to behave deterministically (one transition for every combination of input and state), however during execution some level of non-determinism appears, as it is not certain to state, after how many transitions the machine will terminate for a given input, as the input changes dynamically during the execution. Therefore, having an input tape completely filled up (no blank spaces), it is certain that after some number of moves, eventually blank input will be generated dynamically, what will provide correct input on which the program can halt (even if it hasn’t been provided in the initial input). It is just a matter of time and transitions after which this machine will halt for every provided input. Also, problem
with infinite “vertical” or “horizontal” looping does not appear here (the sequence of
tiles that creates the loop can be dynamically changed at any time during the execution,
what will break the loop).

This algorithm seems to be the slowest one from all of the mentioned ones, i.e. it
usually requires the greatest number of moves to halt the program, but it is not a
general rule. For some inputs it can be faster than the other ones, for some other inputs
it can be slower. However, the greatest advantage of this one is this, it will always
terminate in normal fashion, no matter what input will be provided.

• Running two machines simultaneously

6. Running two basic algorithm machines provides some level of non-determinism,
even though those two machines are implemented as deterministic ones. Running them
simultaneously creates some kind of simulation of “Dynamic Tiles”, because after each
transition for one machine, different tile is being changed as well (result of second
machine working on the same input). It is not possible to predict easily when both of the
machines will halt or will they halt at all.

From the experiments performed, comes out that in about 90% of cases when
input contains at least one blank space, both machines will halt after some number of
transitions. Very rarely they both halt at the same time, usually one of the machines
halts well before the second one does. The interesting thing to notice though is this that
it is possible that one of the machines will halt, but the other one will catch itself into an
infinite loop (“vertical” or “horizontal” one - as mentioned couple of times before). Also,
it is possible that both machines will loop (possibly at the same column or row), however
this has not occurred while performing the experiments.

For the input where no blank spaces occur, both machines will never halt, as they
won’t be able to find the terminating input on the tape. The interesting fact is this that
during the execution, after some period of time, both machines will catch themselves in
an infinite “vertical” or “horizontal” loop. For all tests performed both of the machines
ended up in the same loop, i.e. they were looping on the same column or on the same
row.

All in all, those experiments show, how two Turing Machines, using almost the
same logic can be different from each other and specifically how determinism and non-
determinism affects output result. According to those experiments on the particular
implementations of the 2D Turing Machines, it seems that non-deterministic machines
for some given inputs are less likely to get into an infinite loop; therefore they more
often terminate in normal, regular fashion. Also, to provide some level of non-
determinism, the machine does not have to be strictly implemented as non-
deterministic one, dynamic changes to the input or running multiple machines on the
same input can change the behaviour of the deterministic machine as it would simulate
the non-deterministic one.
Appendix

Source code of the program and used images are included in the “TuringMachine” folder.