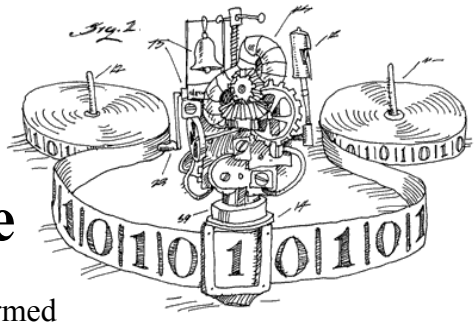
Mathematical Insights in Computing, Junction 2014



Week 2 Challenge Problem:

Programming a Turing Machine

In 1936 Alan Turing described the "a-machine", a device that performed computations on a string of symbols according to a simple set of manipulation rules enumerated in the form of a finite state machine (FSM). The machine consisted of

> ...an unlimited memory capacity obtained in the form of an infinite tape marked out into squares, on each of which a symbol could be printed. At any moment there is one symbol in the machine; it is called the scanned symbol. The machine can alter the scanned symbol and its behavior is in part determined by that symbol, but the symbols on the tape elsewhere do not affect the behavior of the machine. However, the tape can be moved back and forth through the machine, this being one of the elementary operations of the machine. Any symbol on the tape may therefore eventually have an innings. (Turing, 1948)

We now refer to this device as a Turing Machine (TM) and use it as our definition for what it means to be *computable.* Under the Church-Turing Thesis, we believe that a function is algorithmically computable if and only if it's computable by a TM.

The goal of this lab is write the FSM controller for a TM that checks to see if the string of left and right parentheses it finds on its input tape "balance".

The TM has a doubly-infinite tape with discrete symbol positions (cells) each of which contains one of a finite set of symbols. The control FSM has one input: the symbol found in the current cell . The FSM produces several outputs: the symbol to be written into the current cell and a motion control that determines how the head should move. In our simulation, the tape is displayed horizontally, so the tape can move left, right, or stay where it is.

The operation of the TM is specified by a file containing one or more of the following statements:

`// comment`

> C++-style comment: ignore characters starting with the '//' and continuing to the end of the current line.

`/* .... */`

> C-style comment: ignore characters between "/*" and "*/". Note that the ignored characters may include newlines; this type of comment can be used to comment-out multiple lines of your file.

`symbols` *`symbol…`*

> Declare one or more tape symbols. The symbol "–" (dash) is predefined and is used to indicate that a tape cell is blank. You have to declare symbols you use in an action

statement (see below). A symbol can be any sequence of non-whitespace characters not including "/" or the quote character. If you want to declare a symbol containing whitespace, "/" or quote, you must enclose the symbol in quotes. You can have more than one `symbols` statement in your file.

states *state…*

Declare one or more states. There are two predefined states: "*halt*" and "*error*". The TM simulation will stop if either of these states is reached. The "*error*" state is useful for indicating that the TM has halted due to an unexpected condition. You can have more than one `states` statement in your file. The first state specified by the first `states` statement is the starting state for the TM.

action *state symbol newstate writesymbol motion*

Specify the action performed by the TM when the current state is *state* and the current symbol is *symbol*. First the TM will write *writesymbol* into the current cell of the tape. Then the tape is moved left if "l" is specified for the motion, right if "r" is specified and remain where it is if "–" is specified. Finally the current state of the control FSM is changed to *newstate* and the TM searches for the next applicable action. If *newstate* is "*halt*" or "*error*", the TM simulation stops. If there is no action specified for the current state and current symbol, the TM enters the "*error*" state. Note that you have to declare any symbols or states you use in an action statement – this requirement is helpful in catching typos.

tape *name symbol…*

Specifies the initial configuration of a TM tape, each tape has a name. The various names are displayed as a set of radio buttons at the top of the TM animation – you can select which tape is loaded at reset by clicking on one of the buttons. You can specify which cell of the tape is to be current cell after reset by enclosing the appropriate symbol in square brackets. For example, an initial tape configuration called "test" consisting of three non-blank cells with the head positioned over the middle cell is specified by

```
tape test 1 [2] 3
```

If no initial head position is specified, the head is positioned over the leftmost symbol on the tape.

result *name symbol…*

Specifies the expected head position and contents of the tape after the TM has finished processing the initial tape configuration called *name*. This statement is used by the checkoff system to verify that your TM has correctly processed each of the test tapes. Whenever the TM enters the "*halt*" state, the final tape configuration is checked against the appropriate result statement if one has been specified and any discrepancies will be reported in the status display at the bottom of the TMSim window.

result1 *name symbol*

Like `result` except that only the current symbol is checked against the specified value.

Here's an example file that defines a control FSM with three states (s1, s2 and s3) and two possible tape symbols: "1" and "–" (recall that the "–" symbol is predefined). There is a single tape configuration defined called "test" which consists of a blank tape. The final tape configuration is expected to be a tape containing six consecutive "1" symbols with the head positioned over the second "1". Note that there is an action declared for each possible combination of the three states and two tape symbols.

```
// 3-state busy beaver Turing Machine example
// See how many 1's we can write on a blank tape using
// only a three-state Turing Machine

states s1 s2 s3 // list of state names, first is starting state

symbols 1 // list of symbols (- is blank cell)

tape test - // initial tape contents, blank in this case

result test 1 [1] 1 1 1 1 // expected result

// specify transistions: action state symbol state' write move
// state = the current state of the FSM
// symbol = the symbol read from the current cell
// state' = state on the next cycle
// write = symbol to be written into the current cell
// move = tape movement ("l"=left, "r"=right, "-"=stay put)

action s1 - s2 1 r
action s1 1 s3 1 l
action s2 - s1 1 l
action s2 1 s2 1 r
action s3 - s2 1 l
action s3 1 *halt* 1 r
```

To run TMSim, download **tmsim.jar** from http://csvoss.scripts.mit.edu/junction/tmsim.jar. You will need to install Java. Alternatively, log into Athena –

username: espuser
password: (email me to ask)

– and run this in the terminal:

tmsim

Please email me for help if you encounter any problems while trying to run TMSim. It is likely that other students would encounter the same problems as well, and the quicker these problems are fixed for everyone, the better!

After TMSim starts, you'll see a window with the FSM displayed at the bottom and a state/tape animation at the top. The TM display (see figure below) consists of the following parts:

Tape select radio buttons. Select which of the named test tapes to use when initializing the TM after reset.

Tape display: Shows the current state and symbol.

Speed control. This slider controls the speed of the animation when you press the "Run" button. At the fastest speed, no status updates are performed which leads to a much faster simulation.

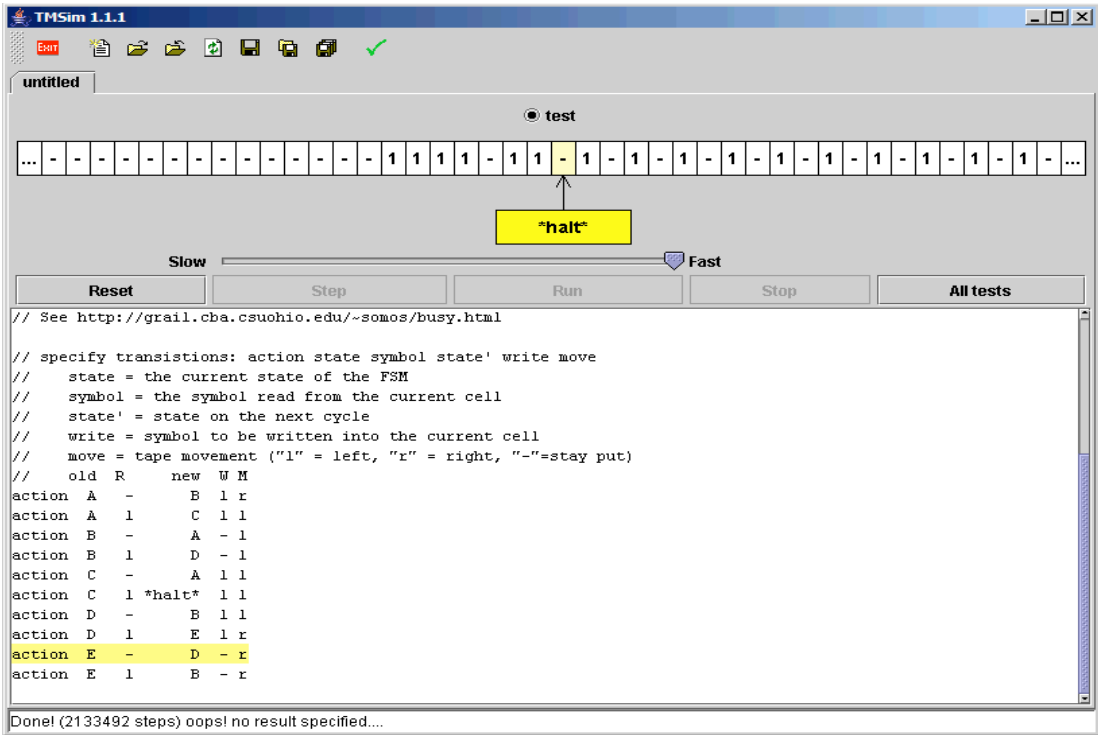
"Reset" button. Reset the TM to its initial state and selected tape configuration.

"Step" button. Let the TM progress one state of the FSM.

"Run" button. Like "Step" except the TM will continue running the FSM until it reaches the "*halt*" or "*error*" state, the "Stop" button is pressed, or an error is detected.

"Stop" button. Stop the TM. You can proceed by pressing the "Step" or "Run" button.

"All tests" button. Automates the task of selecting each test tape in turn and clicking the "Run" button. The automated process will stop if an error is detected.



At the bottom of the screen is a state display showing the current cycle count and any discrepancies detected in the final tape configuration when the TM enters the "*halt*" state.

Task: Parenthesis Checker

Your task is to write the control FSM for a TM that determines if the parenthesis string on its input tape is balanced. Your TM should halt with a current symbol of "1" if the parens balance, or a current symbol of "0" if the parens don't balance. The head should be positioned over the "0" or "1" on the tape. Note that there are no constraints on what the rest of the tape contains.

Here are the test tapes and the expected results. These statements can be found in http://csvoss.scripts.mit.edu/junction/parens_tm.txt and should be copied into the front of your TM file. You'll need to add statements declaring your states and actions (and possibly more symbols) in order to comple te the TM definition.

```
// Parenthesis matcher Turing Machine
// Test tapes! These will help you check your work.
tape test1 (
result1 test1 0
tape test2 )
result1 test2 0
tape test3 ( )
result1 test3 1
tape test4 ) (
result1 test4 0
tape test5 ( ( ( )
result1 test5 0
tape test6 ( ) ( ( ( ) ( ( ( ) ) ( ) ) )
result1 test6 0
tape test7 ( ( ( ( ( ( ) ( ( ) ) ) ) ) ) )
result1 test7 1
// Turing machine definition!
symbols ( ) 0 1
// define your states, actions, and any extra symbols here.
```

You're welcome to add your own test tapes while debugging your implementation.

Points and Scoring

The number of points you'll receive is determined by the number of states in your TM definition. The fewer states you use, the better! Note that you may use *as many symbols as you want* in the alphabet of your Turing machine.

5 or more states:	Full-credit
4 states:	Full-credit + 1 Junction Cup Point
3 states:	Full-credit + 2 Junction Cup Points
2 states:	Full-credit + 3 Junction Cup Points

Turning It In

Either (a) email it to me (csvoss@mit.edu), or (b) pass in a paper copy on Thursday.