CSE 30151 Theory of Computing
Fall 2017
Project 3: K-tape Turing Machine
Version 2: Revised for Individual Work

1 Overview

The goal of this project is to have each student understand at a deep level the functioning of a Turing Machine (TM), how "programs" for such a machine should be written, and what we mean by "complexity" of such programs when executed on real inputs.

Completion of this project will involve two parts. First is writing and running a program that, when given one file that represents a description of a TM, will execute that program against a variety of problems found in a second file representing initial tape values.

The second part is writing programs that demonstrate the operation of a TM in two ways: first as a language recognizer and second as a computation that leaves behind on its tapes the result of a computation.

We are not after world-class performance here, just some relatively simple code that is functional and from which you can draw insight. You are free to go beyond the minimal requirements and produce enhanced code, and if in the instructor's view it truly represents valuable extensions, then it will be considered for extra credit.

You are free to use C, C++, or Python. The latter in particular has proven in the past as perhaps the most efficient way to get decent working code (the overall goal). If you use Python, feel free to use the time, csv, sys, and copy, itertools modules. Use of any other modules requires preapproval of the instructor.

2 The Turing Machine

The main program to be written is a k-tape TM simulator whose operation is as defined in the text. This program, to be called **tm-netid**, where **netid** is your Notre Dame netid, must be written independently by each student, and must take the following inputs (in this order):

- An input machine file defining the TM to be simulated
- An input tape file providing possibly multiple problems consisting of character strings to be run one at a time as the initial contents of the machine's tapes.

2.1 Optimizations

To simplify programming of k-tape TMs, each simulator should implement a few optimizations in its TM operations:

1. The machine file will include a value for "k" that indicates the number of tapes to be used by the machine.
2. The machine file will also include specification of a maximum tape length and a maximum number of steps to simulate. If the execution of a problem exceeds either, the simulator should halt with an appropriate error message.
3. When reading a new problem from the tape file, a k-tape machine will read k lines, one to initialize each of the k tapes. This saves coding to place some start-up information on the tapes other than the first. If the input line for some tape has fewer characters than the maximum tape length, the rest of the tape positions to the right of the provided string are all to be considered blank.
4. The special character "*" cannot be part of either Σ or any of the k Γs, and if used in place of a character from Γ in the left-hand side of a transition rule will signal that it matches any character on the specified tape. This reduces the number of rules that have to be written, primarily for multi-tape problems.
5. The same special character "*" on the right-hand side of a rule says that the character under that tape's head should be left alone and not changed. This is useful to again avoid multiple rules just to cover a case where changing the tape character is not wanted.
6. Besides "L" and "R" for head direction, an "S" option is allowed that says "stay", i.e. don't move that tape head. For $k > 1$ this permits some tape heads to be left alone while others are moved.

2.2 Project Requirements

All students will complete the same requirements. Your TM simulator must be able to simulate a TM with an arbitrary number of tapes. The number of tapes that are to be simulated is a parameter that will be read in from the machine file.

2.3 The Simulator in Operation

When the TM simulator is activated, the specified machine file containing the TM's program should be read in and internally processed to whatever internal representation you are using. As each rule is read in, an integer "Rule #" should be associated with it, assigned in sequential order. The Rule #, followed by a ":", and an echo of the rule should be dumped to stdout so that later traces can be followed. Part of the machine file defines "k" - how many tapes the machine should have.

After reading in the program file, the name of the tape file shall be echoed to stdout. Then before each new problem in the tape file, the TM should be reset to its start state, all statistics should be cleared, all tapes cleared to all blanks, and the next k lines from the tape file copied into the TM's tape(s). The TM should then be allowed to run until either:

- The TM enters either the accept or reject states,
- A tape character is read that is not part of that tape's Γ alphabet,

- There is no rule covering the current state and input character (i.e. a transition to a trap state).
- The TM takes more steps than is set as a parameter in the machine file.

After each run, if there are more input problems (as signified by more lines in the tape file), the TM is reset and execution repeated on the next set of k strings, using the same TM program.

3 Test Files

Test files are available at:

`https://www3.nd.edu/~kogge/courses/cse30151-fa17/Public/Projects/Project3/TestFiles/`

Within this directory there are several test files associated with tm-netid:

- Four sample machine files: TM1d.txt, TM1.txt, TM2.txt, and TM3.txt. The first is a single tape language "d"ecider; the last three may be "computation" machines that leave some result on a tape. A machine with prefix "TMk" requires k tapes.
- For the TM1d machine there are two tape files with suffixes "-accept" and "-reject" containing strings that represent strings that are, or are not, part of the language accepted by TM1d.
- For each of the other machines is a tape file with prefix "tape-" that contains a set of test strings for the associated machine. The answers for these machines is not given.

All students must run all machines and all problems on their TM simulator.

3.1 Student-supplied Machines

In addition to instructor-supplied machines and test programs, each student shall create two separate and different machine and matching test strings for their TM. These should be documented in the readme and included with the code. One of these shall be a decider for some language; the other can be either a decider or a computation. At least one of your machines must use more than one tape.

The book and/or homework problems may be used for inspiration of the problems to be solved. The conversion of a problem into a machine file and associated test tape files must be your own independent work.

The name field of each machine should include your netid.

Note that many problems in the book append some special character on the leftmost position of the tape, and shift all the tape characters to the right before starting the computation. Feel free to add such special characters to the initial tape to begin with to simplify the machine design.

4 Documentation

Several types of documentation, all in PDF format, are required. A readme-netid.pdf and separate reports entitled machinename-netid.pdf are to be submitted for each student-designed machine.

4.1 readme-netid

A key part of what you submit is a readme-netid.pdf that includes the following in this order:

1. Your name and netid.
2. Approximately how much time was spent in total on the project.
3. A description of how you managed the code development and testing. Use of github in particular is a strong suggestion to simplifying this process.
4. The language you used, and a list of libraries you invoked.
5. A description of the key data structures you used, especially for the internal representation of tapes, states, and the state machines and the transitions.
6. If you did any extra programs, or attempted any extra test cases, describe them separately.

4.2 Individual Machines

Each student shall submit in their Sakai directory a copy of the two machines that they designed and ran, the test files they created, and trace files of their execution. Also included shall be a brief write-up, in a .pdf file of:

1. What problem the machine tackled. Was it a recognizer, decider, or computation problem.
2. What, if any, was the reference from which the problem solved by the machine was drawn.
3. How does the machine work, in words.
4. A state diagram.
5. How did you verify correct operation.

5 File Formats

5.1 Machine Format

The file provided with the rules for the TM shall consist of a file where each line provides distinct information:

- **Line 1:** A comma separated line of the following
  - The name of the TM machine.
  - The number of tapes needed by this machine.
  - The maximum tape length
  - The maximum number of steps
- **Line 2:** The Σ alphabet to be used for the initial main tape: only single ASCII letters are allowed, comma separated, as in "a,b,c,...". Any ASCII character other than "*" or "_" is allowed. "*" is reserved for use as a "wildcard" in transition rules as discussed above in Section 2.1, and "_" (underscore) stands for a blank.
- **Line 3:** The names of the states, separated by commas, as in q0,q1,... There is no constraint on the length or character set of a state name.

- **Line 4:** The name of the state that should be considered the start state.
- **Line 5:** The name of the accepting state and rejecting state, separated by a comma.
- **Line 6 thru 5+k:** The set of symbols to be used for the Γ for each tape. "*" cannot be used, and "␣" is automatically included (you do not need to add).
- **Lines beyond:** one transition rule per line, in a comma-separated format:

  Initial State Name{, *Input Symbol*,}^{k} *New State Name*{, *New Symbol*}^{k}{, *Direction*}^{k}

  The "k'th" position in any list of length k corresponds to the k'th tape.

  The Input Symbol and New Symbol is any symbol from the Γ for that tape, or a "*" or a "␣" (blank).

  Each of the k Direction symbols may be from "L" (left), "R" (right), or "S" (stay).

  Processing of rules stops with the end of file.

The comma-separated format should be compatible with a ".csv" form, allowing a machine description to be prepared easily by a spreadsheet such as excel if desired. In fact a valid extension for such files could be ".csv."

Note that it may be that such files are produced on a Windows machine that adds a carriage return and a linefeed to the end of each line.

5.2 Test File Format

Each test file is a text file, with a set of k lines representing the initial tape contents of the k tapes, and characters for tape 1 only from Σ, and from tapes 2-k from Γ_i. Note that it may be that such files are produced on a Windows machine that adds a carriage return and a linefeed to the end of each line.

5.3 Output Format

After processing the rules file, the output from the execution for each problem set of k lines run shall be sent to stdout and consist of:

- The initial k tape contents, with each line prefixed by "Tape i:" Blanks at the right of the initial values need not be shown
- For each transition, the following on a separate line:

  Step number, *Rule Number*{, *Tape Index*}^{k}, *Initial State Name* {, *Input Symbol*}^{k}, *New State Name*{, *New Tape Symbol*}^{k}{, *Direction*}^{k}

  The Tape Index are integers representing where the head on each tape is at the start of the transition, in 0-origin so that the leftmost tape position is "0."
- After all transitions have been performed, print either "Accepted" (if the last state was the accepting state), "Rejected" (if the last state was the rejecting state), or "Error" (otherwise).
- The final contents of the k tapes, one per line, with a prefix of "Tape i:" on each line.

The comma-separated format should be compatible with a ".csv" form, allowing a redirect to a ".csv" file so that the output can be read by a spreadsheet program such as excel.

6 Submission

- Each student should have in their Sakai directory for this course a directory called Project3, where all submissions should go.
- When you are ready to submit, place copies of all code and test machine output in the designated directory.
  - Your code should be runnable on any of the studentnn.cse.nd.edu machines so that if there is an issue the graders can run the code themselves.
  - If you wrote in a compiled language like C++, include all needed source files (excepting standard libraries), a make file, and a compiled executable. The source code is there to allow the graders to look at the code for comments and to resolve any discrepancies that may arise in looking at your results.
  - If you wrote in a language like Python make sure your code is compatible with one of the versions supported on the studentnn.cse.nd.edu machines, again to allow the graders to check something if there is an issue.
- Also in the same directory as the code you should place an output file for each of the test files that you ran. The format should be as described in Section 5.3, and the name should be the same name as the test file but with a "results-" prefix on the name.

In addition, you should include in your Project3 Sakai directory:

- A copy of your readme-netid.pdf file.
- Machine and test files for the designs that you created.
- The output files from running the above machine files against the string files.
- A short readme on each machine as described in Section 4.2.

7 Grading

Grading of the project will be based on 100 points, divided up as the following:

- Points off for late submissions (10 points per day).
- 5 points for following naming and submission conventions.
- 10 points for "reasonably" commented source code.
- 40 points based on the percent of cases you got correct for running the provided test problems.
- 25 points for completeness and quality of the readme file.
- 20 points for the student-designed machines and their tests.