CSE 30151 Theory of Computing
Spring 2018
Project 3: K-tape Turing Machine

Version 1: March 21, 2018

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 descsription 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 modes: first as a **language recognizer** and second as a **computation engine** that leaves behind on its tapes the result of a computation.

It is expected that much of the DFA simulator written in the second project can be re-used here. 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.

You are also free (i.e. "encouraged") to implement your TM on your Arduino.

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-*team***, where *team* is the name of your team, must be written independently by each group, and must take the following inputs (in this order):

- An input **Machine file** defining the TM to be simulated
- An input **Initial 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 is blank or 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 Complexity

To try to make the workload per student approximately equal, the number of students in a group will determine the maximum number of tapes that the group's TM must be able to handle, as follows:

- If the group has 3 students, their TM 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.
- If the group has 2 students, their TM need only be capable of simulating 1 or 2-tape TMs. Again the number of tapes that are to be simulated is a parameter that will be read in from the machine file. A machine file specifying more than 2 tapes should be rejected.
- If the group is a single student, only TMs needing a single tape need be handled. If the machine file specifies more than 1 tape, it may be rejected.

2.2.1 Arduino TMs

If an Arduino project is attempted, as with the DFA project, a group of 2 students can collaborate on implementing just a single tape TM. The reason is as before - the complexity of transferring Machine and Initial Tape files between the host and the Arduino.

Likewise a 3 person team could implement a 1 or 2-tape TM on an Arduino.

A single person team attempting a 1-tape TM on an Arduino would get extra credit, as would a 2 person team that attempts a k-tape machine.

The instructor has a number of the small 128x32 Blue/yellow dot displays that could be used to display the tapes, especially if the alphabet is limited to at most 4 characters {Blue, Yellow, Green (both), blank}.

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 the group is using. As with the FA simulator, 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 (as described in Section 5.1) defines "k" - how many tapes the machine should have.

After reading in the program file, the name of the initial 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.

Note that as in the DFA project your design should automatically assume a "TRAP" state to which all undefined transitions go.

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

Under the Projects tab of the class website https://www3.nd.edu/ kogge/courses/cse30151-fa17/index.html there is a directory called Project3. Within this directory there are several test files associated with **tm-*team***:

- 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.

A group of 3 students must run all machines and all problems on their TM simulator. A group of 2 students need not run the TM3 machine. A group of 1 student need run only the two 1-tape machines.

3.1 Student-supplied Machines

In addition to instructor-supplied machines and test program, each individual student shall create two separate and different machine and matching test strings for their group's 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. For multi-tape groups, at least one of each student's selections must use more than one tape. Thus, if there are 3 students in a group, then 6 different machines and test code shall be included.

The book and/or homework problems may be used for inspiration of the problems to be solved. Group discussion of interesting problems is encouraged, but the conversion of a problem into a machine file and associated test tape files must be the work of each individual student. Different students in the same group should design different machines.

The name field of each machine should include the *netid* of the student doing the design.

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. One (entitled **readme-*team*.pdf** is submitted once by the entire team; a separate **teamwork-*netid*.pdf** is submitted separately by each member of the team. Finally, separate reports entitled **machine-name-*netid*.pdf** are to be submitted for each individual student-designed machine.

4.1 readme-team

A key part of what your teams submit is a **readme-*team*.pdf** that includes the following in this order:

1. The members of the team.

2. Approximately how much time was spent in total on the project, and how much by each student.

3. A description of how you managed the code development and testing. Use of github in particular is a particularly strong suggestion to simplifying this process, as is some sort of organized code review.

4. The language you used, and a list of libraries you invoked.

5. If you built upon some code from the FA project (highly encouraged), the name of the FA project team whose code you built on.

6. A description of the key data structures you used, especially for the internal representation of tapes, states, and the state machines and the transitions.

7. If you did any extra programs, or attempted any extra test cases, describe them separately.

Only one member of the team need submit this report to their Sakai project directory, along with the TM code.

4.2 Individual Machines

Each student shall submit in their own Sakai directory a copy of the two machines that they personally 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.

4.3 teamwork-netid

In addition, each team member should prepare a brief discussion of their own personal view of the team dynamics, and stored in a PDF called **teamwork-*netid*.pdf**, where netid is your netid. The contents should include:

1. Who were the other team members.

2. Under whose netid is the **readme-*team*.pdf**, code, and other material saved.

3. How much time did you personally spend on the project, and what did you do?

4. What did you personally learn from the project, both about the topic, above programming and code development techniques, and about algorithms.

5. In your own words, how did the team dynamics work? What could be improved? (e.g. did you use github and if so did it help, did you meet frequently enough, etc.)
6. From your own perspective, what was the role of each team member, and did any member exceed expectations, or vice versa.

Each student's submission here should be in their own words and SHOULD NOT be a copy of any other team members submission, nor should they be shared with the other team members. These reports will be kept private by the graders and instructor, and will be used to ensure healthy team dynamics. The instructor retains the right to adjust the score of an individual team member from the base score (both up and down) on the basis of these reports. Also, a composite of all such reports from all projects will be used to create an overall "lessons learned" at the end of the project in what techniques seemed to work better, and where problems arose. These hopefully will be of use for the next project.

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).

As with the DFA project, it is not mandatory that you use all this information. For example you may choose to ignore the line of state names if your machine does a search each time anyway for the matching rule.

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 Initial Tape 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.

5.3.1 Arduino Implementation

If you implement an Arduino-based TM you may want to do something other than write to stdout thru the host, for example use a display connected to the Arduino. In this case simly describe in the writeup what you should see, and then demo it to the instructor (I am waiting with bated breath to see lots of these!).

6 Submission

- Each team member should have in their own Sakai directory for this course a directory called Project3, where all submissions should go.
- When the team is ready to submit, one (and only one) team member will place copies of all code and test machine output in the designated common directory.
  - In not an Arduino implementation, 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 the team 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, every team member should include in their own Project3 Sakai directory:

- A copy of their individual **readme-*team*** file, again in pdf.
- Machine and test files for the designs that the student did themselves.
- The output files from running the above machine files against the string files.
- A short readme on the machine as described above.

7 Grading

Grading of the project will be based on a 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 problem.
- 20 points for completeness and quality of the readme file.
- 20 points for the student-designed machines and their tests.
- 5 points for the teamwork report.

All but the last two items will be common to all members of a team. The last two are specific to each member.