PJ03 – DTM Emulator

In this project, you will implement an emulator for a deterministic, single-tape Turing machine. This is very similar to the machine you implemented in PJ01 and, in fact, you should be able to make fairly minor modifications to that emulator to implement this one.

Similar to prior projects, emulating a Turning machine determines whether or not the input string is in the language recognized by the machine being emulated and echoing out to an output file those strings from the input file that are in the language.

Our machine description files are actually a bit simpler than those for earlier machines. Since a Turing machine has defined Accept and Reject states, there is no need to have the first line give this information. We will use State 255 as the Reject State and State 254 as the Accept State. As before, State 0 is the start state.

The machine description file contains only the transition definitions. These are of the form

FromState, ReadSymbol, ToState, WriteSymbol, HeadDirection

The FromState and ToState fields are 8-bit unsigned integers.

The ReadSymbol and WriteSymbol are a bit more complicated. Each will be of the form c<n>, meaning a one-character symbol (which must be a printable character) optionally followed by a numeric modifier. More about this later.

The HeadDirection will use the characters ‘L’ and ‘R’ for the two allowed movements.

It is not uncommon for the tape alphabet to be significantly larger than the input alphabet with multiple versions of each input symbol – for instance, each input alphabet symbol with one dot, with two dots, with a hat, with a hat and two dots, etc. – to aid in computation. These additional marks are often referred to as “decorations”. While there is no requirement for the encoding of a decorated symbol to bear any relation to the undecorated symbol, it generally makes things easier if there is a simple relationship. To facilitate such an implementation, the tape symbols can be decorated by appending an 8-bit integer to them. For instance, the transition

13, t, 18, t3, L

Says that if the machine is in State 13 and a ‘t’ is under the tape head, then the machine replaces the ‘t’ with a ‘t’ decorated with a 3 (perhaps that means a “hat”). This should be encoded as a single symbol on the tape and an easy way to do that is to use a two-byte tape alphabet in which the upper byte is the 8-bit value of the decoration and the lower byte is the 8-bit ASCII value of the basic symbol. Using this strategy, t and t0 would be the same symbol.

Before, the input alphabet was the set of printing characters. But we need the ability to define, using printing characters, tape symbols (such as the blank symbols) that are not in the input alphabet – what we will call tape-only symbols. Therefore, we will remove one printing character, namely the '@' symbol, and reserve it to indicate such a tape-only symbol. The blank symbol will be represented by @0. All other tape-only symbol definitions are up to you.

How you encode the tape-only symbols is also up to you. One option is to encode them exactly like other symbols, which would mean that a blank symbol on the tape would be '@'. You might also choose to encode the '@' portion of the symbol (i.e., the lower byte) as a zero, which would make the blank symbol encoded on the tape as zero. This is completely up to you.

You will be supplied with some machine definition files (using the naming convention m00.tm where the 00 is replaced with the machine number) and you will run these on the supplied input file. Your output will again be a text file (named m00.txt for the m00.tm machine file) containing the output from each machine.

You will need to generate a single log file, named tm.log, summarizing the results of running each of the machine descriptions. This log file only needs to contain the base name of the machine followed by the number of strings that were accepted (separated by a comma) with one machine per line. For example:

```
M00,4521
M01,947
```

SUBMISSION

You need to submit a single ZIP file. At the top level should be a single-spaced report (in PDF format) that is no more than five pages long (two to three is preferred). This report should focus on describing the technical approach taken in implementing your emulator.

Also include a short section indicating the programming language and/or development environment that you used and any special configuration settings needed by someone that would like to reproduce your program as well as how to run your program.

There should also be two folders at the top level. The first is named "results" and should contain your output .txt files as well as the log file. It does not need to have any other files, including the original .tm files, but it may contain them. There should also be a folder named "code" that contains all of the source code for your program. Included should be all files needed by someone to reproduce your program assuming they have the appropriate development environment. This folder can contain subfolders as appropriate.

Keep in mind that the grader is very unlikely to even attempt to run your program – they likely will not know the language you used, let alone have the necessary tools. Thus it is important that your report file describe your approach and algorithm very clearly AND that your code be well organized and commented – the grader WILL review your code.

GRADING CRITERIA

Report/Code: 50%

- 70% Technical merit – algorithm is well explained.
- 10% Programming environment details well explained.
- 10% Code is well organized
- 10% Code is well documented

Output Files: 50%

Each file will count equally toward this portion of the grade.

For EACH .txt file:

- -1% for each string acceptance error (either falsely accepted or falsely rejected).