Assignment 10 - Turing Machines

CS 234

due May 5th, 11:59pm

0 Introduction

This assignment is to be completed individually, but feel free to collaborate according to the course's external collaboration policy (which can be found in the syllabus). *Generative AI usage must follow course guidelines to be eligible for points.*

The deliverables consist of one `.pdf` file, one `.py` file, and 2 `.yaml` files from turingmachine.io. The deliverables should be submitted electronically to by the deadline. Put any attribution text in the `.pdf` file. You may also consider adding an experience report to the `.pdf` describing your experience with the assignment: how long did it take, how hard/fulfilling was it, etc.

Your `.pdf` file should be named like `FLast_cs234_aX.ext` where `F` is your first initial, `Last` is your last name, `X` is the assignment number, and `ext` is the appropriate file extension. For example, Alan Turing's `.pdf` file should be given the name `ATuring_cs234_a10.pdf`. (Alan Turing invented the Turing machine, and that kicked off the field of computer science!)

1 Part 1 – Configuring Turing Machines

Please complete the following exercises from the textbook on turingmachine.io. Look at the examples to see the syntax. The basic idea is that a transition from state `state1` is given by the following syntax written below the `state1:` header: `A: {write: 'B', R: state2 }` . This transition indicates that, when reading an `A`, the Turing machine will write `B`, move the head right (`R`) and transition the Turing machine to the `state2`.

Files can be downloaded by going to the share button in the upper right and clicking "download document". Each file should be named like `FLAST_NUMBER.yaml` where `F` is your first initial, `LAST` is your last name (no spaces), and `NUMBER` is the textbook's task number (not counting the chapter). For example, Alan Turing's submission to 13.14 would be named `ATuring_14.yaml`.

- 13.14
- 13.19

2 Part 2 – Reducing with Turing Machines

Please complete the following exercises from the textbook in your `.pdf` submission. Clearly label your responses with the exercise number.

- 14.10
- 14.12
- 14.13
- 14.14

3 Part 3 – Simulating a Turing Machine in Code

For this part of the assignment, you are to code two functions to simulate Turing machines. To help with this task, a stub file has been provided on Canvas: `a10.py`. You are to edit this file (without renaming it) to create your solutions. This stub file contains some example Turing machines, as well as the helper function `removeTrailingBlanks` which you may use to help implement your functions.

For this coding, you will need to represent the configuration of a Turing machine. However, there is no data structure immediately suitable to representing the Turing machine's tape, which goes infinitely in both directions. Thus, we instead represent a such a tape with 2 lists: one for negative indices (before the input string location), and one for nonnegative indices. Indices beyond these lists will be assumed to hold blank characters. Each Turing machine configuration can then be represented with the following 4-tuple $\langle q, negTape, tape, z\rangle$. In this tuple:

- q is the current state—either something from the formal set of states Q or the special reserved string `"REJECT"`. For convenience, we reserve the string `"REJECT"` for representing the implicit rejection state reached when no transition is otherwise available. (Note that, the way we have defined Turing machines, the state `"REJECT"` is not actually an element of the formal set of states Q.)

- $negTape$ is a list of symbols representing the (reversed) tape contents before the input string, where negative tape indices would be. Tape index $-i$ corresponds to $negTape$ index $(i-1)$. Thus, if $negTape = [c, a, t]$, then the tape contents before the input string contains infinitely many blanks followed by t, a, c, and the input string would have originally been placed after the c.

- *tape* is a list of symbols representing the tape contents starting with the input string. The first index of *tape* is 0, and that is where the first letter of the input string goes.

- neither *tape* nor $negTape$ have trailing blank symbols. This keeps each representation of a configuration unique. If a Turing machine would read a symbol at an index past the contents of *tape* and $negTape$, it simply assumed to be the blank symbol. If a Turing machine would write a symbol at an index past the contents of *tape* and $negTape$, then however many blank symbols are needed should be added to the end of the appropriate list, followed by the desired symbol to be written.

- z is an integer representing the index of the tape that the Turing machine is about to read

Function 1 The first function you need to code is `stepTM(tm, config)`, which takes a Turing machine tuple `tm` and a current configuration `config`.

The simulator then returns the Turing machine configuration reached after 1 more step of the machine. (This function is allowed to mutate `config`.)

- `tm` is a 7-tuple $\langle Q, \Sigma, \Gamma, \delta, q_0, B, F\rangle$ where:
  - Q is a set of elements representing states (and `"REJECT"` $\notin Q$)
  - Σ is a set of length-1 strings
  - Γ is a set of length-1 strings, and $\Gamma \supseteq \Sigma$
  - δ is a set of 2-tuples. The left element of each 2-tuple is an element of $Q \times \Gamma$, and the right element is an element of $Q \times \Gamma \times \{\text{"L"}, \text{"R"}\}$. Also, each left element is unique, so that δ is a functional relation.
  - q_0 is the start state, and $q_0 \in Q$
  - B is the blank symbol, and $B \in \Gamma$
  - F is the set of accepting states, and $F \subseteq Q$
- `config` is the current configuration of the Turing machine in the 4-tuple format discussed previously. You may assume that this configuration is valid with respect to `tm`, so that, e.g., the tape only contains symbols from Γ.

Note that, when transitioning to a rejecting state (i.e., when no transition exists for the configuration in δ and the state is not already accepting or rejecting), the only part of the configuration that should change is the state, which changes to `"REJECT"`. Additionally, accepting and rejecting configurations do not change once reached, and so are considered to transition to themselves with 1 more step.

Function 2 The second function you need to code is `runTM(tm, inString)` which runs the the Turing machine `tm` (the same sort of 7-tuple as described previously) on the input string `inString`. The function then returns `True` if the Turing machine eventually reaches an accept state, returns `False` if it eventually reaches a reject state, and runs forever if the Turing machine runs forever. (You should recognize this behaviour as that of the universal Turing machine.)

You may assume that `inString` contains only letters in the alphabet of `tm`. You may also assume that `tm` is a specification-compliant 7-tuple.