Busy Beaver Lab

Dr S. S. Chandra

A Turing machine is a state-based representation of computation that effectively provides a computer in its simplest form. One of the key components of a Turing machine is the transition rules that encodes and implements an algorithm that the Turing machine can run. See Chapter 7.5 of (Moore and Mertens, 2011) for more details.

In this laboratory, you will create Turing machines based on a Python simulator and program them using transition rules. The Python Turing machine simulator runs with a single sided infinite tape designed using Python generators. You will also be able to print the configuration of the machine during computation to debug and implement the algorithms requested in the following parts of the laboratory.

Part I – Turing Machine Simulator (2 Marks)

A set of example scripts are provided to give programming examples of the Python simulator. Run the test programs below with some different input strings and examine the outputs.

[See scripts `test_turing_machine_example1.py` and `test_turing_machine_example2.py`]

a) List the states for each of the Turing machines in the test scripts provided.

(1/2 Mark)

b) Change the run command in the scripts so that one can observe the configuration of the machine per transition.

(1/2 Mark)

c) Describe what computation each Turing machine example performs.

(1 Mark)

Part II – Turing Machine Arithmetic (5 Marks)

A Turing machine can represent any computation including arithmetic. We will implement some basic addition and multiplication machines before progressing to more general computations.

[See starting code in `test_turing_adder.py` and `test_turing_multiplier.py`]

a) Create a Turing machine that computes the addition of two numbers using unary representation. You may assume that the numbers are separated by '0' and the blank symbol is the empty string. Demonstrate that the machine can compute the following inputs:
   i. 2+3 as 110111
   ii. 3+4 as 11101111

(2 Marks)

b) Create a Turing machine that computes the multiplication of two numbers using unary representation. You may assume that the numbers are separated by '0' and the blank symbol is the empty string and utilize additional tape symbols as needed. Demonstrate that the machine can compute the following inputs:

i. 2*3 as 110111
ii. 3*5 as 111011111

(3 Marks)

Part III – Busy Beaver (8 Marks)

The Busy Beaver game (see links in the appendix) is creating a halting, binary alphabet Turing machine that writes the greatest number of ones on its tape using a limited number of states (or cards) plus an accept or halt state. Each card usually represents a state and its transitions based on the alphabet character encountered. The following is a 2-card program

A	B
0: B1R	0: A1L
1: B1L	1: H1R

[See starting code in `test_busy_beaver_2.py` and `turing_machine.py`]

a) The current implementation of the tape in the simulator is infinite only in the right direction. Modify the turing_machine.py module so that the Turing machine simulator now uses a double sided (left and right sided) infinite tape.

(2 Marks)

b) Why is using Python generators advantageous in simulating Turing machines?

(1 Mark)

c) Program the Turing machine simulator to run the 2-card Busy Beaver program above. How many ones do you get?

(1/2 Mark)

d) Check this result with the known 2-card programs and how does your result compare?

(1/2 Mark)

e) Write your own 3-card and 4-card Busy Beaver programs and demonstrate the number of ones your program writes.

(2 Marks)

f) Can you find a new 5-card Busy Beaver program? Justify why you believe your 5-card program is new and either demonstrate that it halts or indicate reasons for why it should do so.

(2 Marks)

Appendix

YouTube Video - Turing Machine Primer - Computerphile

YouTube Video - Busy Beaver Turing Machines - Computerphile

References

Moore, C., Mertens, S., 2011. The Nature Of Computation. Oxford University Press.