Turing Machine Coding

1 Introduction

In a lot of the high-level descriptions of Turing machines we have discussed, they take as input another Turing machine along with a tape and then the first Turing machine executes the second on the tape. For example, the language

$$A_{TM} = \{\langle M, w\rangle | M \text{ is a TM that accepts } w\},$$

which is not *decidable*, is *recognizable* using the universal Turing machine

$U =$ " On input $\langle M, w\rangle$, where M is a TM and w is a string:

1. Simulate M on w.
2. If M ever enters its accept state, *accept*; If M ever enters its reject state, *reject*."

The objective of this assignment is to mimic what the universal Turing machine is doing by fleshing out a class (see code block 2) which can be initialized with an appropriate transition function and tape. In particular you should:

1. (70%) Complete the Turing machine class (code block 2) so that you can generate output like that found in tables 1a and 1b.
2. (20%) Create a new transition function which extends the function in code block 1 so that it can handle negative numbers. (Hint: Draw the transition diagram first.)
3. (10%) Use the add functions to create a multiply function that can multiply integers together. (Hint: Create a diagram that simulates multiplication using repeated addition.)

If you complete problem 1 correctly then for each of the other two parts you just need to focus on the transition function which is really just a description of an appropriate transition diagram using code. For the functions you may assume that the machine already knows how to add and subtract 1, that we may consider integers enumerated and so available to us, and that it can already compare other integers to 0.

I wrote this up using Python because that was most convenient for me, you are free to work in any language you like. If you want copies of the Python code the *.py files are available as part of the write up for this project https://www.overleaf.com/read/xyyqdyrhsqkf

2 Adding Turing Machine

The Turing machine in figure 1 will add two non-negative integers together.

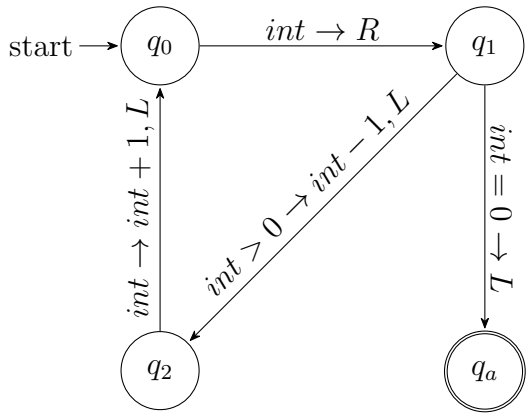

Figure 1: Adding Turing Machine

For example if you input $\langle 3, 2\rangle$ for the tape and run the machine until it halts you get the computational history in table 1a. Where as if you input $\langle 3, 1.7\rangle$ the computational history is in table 1b.

C_0 :	(q0)	3	2
C_1 :	3	(q1)	2
C_2 :	(q2)	3	1
C_3 :	(q0)	4	1
C_4 :	4	(q1)	1
C_5 :	(q2)	4	0
C_6 :	(q0)	5	0
C_7 :	5	(q1)	0
C_8 :	(accept)	5	0

(a) Adding 2 to 3

C0:	(q0)	3	1.7
C1:	3	(q1)	1.7
C2:	(reject)	3	1.7

(b) Adding 3 and 1.7

Table 1: Computational Histories

You can see by the history in table 1a that the machine works by repeatedly decrementing one integer and incrementing the other; the high level description of the machine would be something like this

$A =$"On input of $\langle m, n\rangle$:

1. If m or n is not a non-negative integer, then *reject*.
2. If $n > 0$, decrement n and increment m.
3. Repeat step 2 until $n = 0$, then *accept* and then print m."

An implementation/formal level description of this might be coded in Python as in code block 1.

```python
# Transition function to add two non-negative integers
def add(M):
    if M.state=="q0" and type(M.head())==int: M.move_right("q1")
    elif M.state=="q1" and type(M.head())==int:
        if M.head()>0:
            M.write(TM.head()-1)
            M.move_left("q2")
        elif M.head()==0: M.move_left("accept")
    elif M.state=="q2" and type(M.head())==int:
        M.write(M.head()+1)
        M.move_left("q0")
    elif M.state=="accept": M.move_left("accept")
    else: M.move_left("reject")
```

Code Block 1: Implementation/Formal Description of the Adding Machine

3 Class Shell

To run the adding machine in code block 1 we need a Turing machine (TM) that knows how to move left and right, read the tape value at the position of the head, and overwrite the current value. If we implement the TM as a class, each machine like code block 1 can quickly be simulated on a given tape. In particular, completing the class description in code block 2, we can create an instance of the class `AddTM=TM(add,[3,2],"q0",0)` or `BadAddTM=TM(add,[3,1.7],"q0",0)`, which can produce the computation histories in tables 1a and 1b by repeatedly executing `AddTM.delta()` and printing the result.

```
class TM:
# Initialize the class
    def __init__(self,transition,tape,state="q0",position=0):
        self.transition=transition # Function for particular action
        self.tape=tape # List representing the values on the tape
        self.state=state # State name
        self.position=position # Head position on tape
# Apply the transition function once to TM
    def delta(self):

# Move the head position one right and update the state,
# if head is at the right side add a blank
    def move_right(self,state):

# Move the head position one left and update the state,
# if head is at the left side don't change position
    def move_left(self,state):

# Return the value at the current head position
    def head(self):

# Overwrite the current head position
    def write(self,input):

# Return a string representation of the TM in the format
# "...a<state>b..." where <state> is the current state and
# b is the current value of head
    def str(self):
```

Code Block 2: Shell of a TM Class Definition