CSC445/545 - Exam #3 (Final)

Jim Leon

May 2, 2021

Due: May 3, 2021 2pm MST.

1 Program Description

This program explores and simulates various uses of a single-tape Turing Machine. Depending on the users' selection from the various menus, the program will ask for a 'tape' (a user input string) and will run this tape through a virtual Turing Machine, simulating the moves and changes to the tape along the way. The machine will prompt the user if it either completes the simulation (if an answer was found) or exits the simulation prematurely (encountered a non-described transition function).

This program was written in Python version 3.8.5. The authors machine was running the Linux Ubuntu 20.04.2 LTS operating system on a Intel® Core™ i7-6600U CPU (2.60GHz × 4) at the time of testing and release. This application was not tested on any other machine and makes no guarantees to it's ability to run on other machines or other software configurations. The author makes no warranties of support for this application after this release date.

2 Example of running the application

This is a menu-driven application. As such, the program can be fired up by navigating to the applications main directory and entering the following into a terminal window:

```
$ ./main.py
```

2.1 Main Menu

The program starts by displaying a brief description and author name in the terminal. Directly below this there will be a Main Menu:

```
    THE TURING MACHINE SIMULATOR
============================================================
This program simulates various uses for the Turing Machine.
Author:  Jim Leon


------------------------- Main Menu -------------------------
1.  Language Acceptors
2.  Adders
3.  Multiply & Divide
```

```
4.  Import custom transducer from file
5.  Exit Application

Selection:
```

After the selection prompt, the user can enter the number corresponding with the menu choices. This will direct them either to a sub-menu (in the cases of menu selections 1, 2, and 3), an input command (selection 4), or will end the application (selection 5). Entering numbers or other characters that do not correspond to the menu choices given will result in an error message and the reprinting of the menu:

```
Invalid selection.

------------------------- Main Menu --------------------------
1.  Language Acceptors
2.  Adders
3.  Multiply & Divide
4.  Import custom transducer from file
5.  Exit Application

Selection:
```

The application is intuitive from here. Explanations of what constitutes acceptable input and what sort of constraints are instituted on that input should be described throughout the applications use. Below is a brief overview of the other menu's the user will encounter while using the program.

2.2 Language Acceptors

```
-------------------- Language Acceptors ----------------------
1.  L = {(a^n)(b^n)(c^n), n > 0}
2.  L = {w w^R, |w w^R| is even, 'sigma' = (a,b)}
3.  Import custom language acceptor from file
4.  Back to Main Menu

Selection:
```

If the user selects option 1 from the Main Menu, they will be directed to the Language Acceptors sub-menu. Here they have the ability to explore two specific simulations of a single tape Turing Machine in action by selecting either menu option 1 or menu option 2. By selecting option 3, the user can enter the name of a custom file - *the user must include the relative path of the file!* - from which they can run the simulation.

Selecting option 4 will return the user to the Main Menu.

2.3 Adders

If the user selections menu option 2 from the Main Menu, they are directed to the Adders sub menu:

```
------------------------- Adders ---------------------------
```

```
1.  Add unary numbers (e.g. 111+1111)
2.  Add binary numbers (e.g. 010110+000011)
3.  Back to Main Menu

Selection:
```

Here, again, the user is given two specific simulations they can run that demonstrate a single-tape Turing Machine being used as a unary adder (option 1), or a binary adder (option 2). Menu option number 3 will return to the Main Menu again.

2.4 Multiply & Divide

This is where things get fun! The single-tape Turing Machine can also simulate multiplication and division. By selecting menu option 3 from the Main Menu, the user is brought to the Multiply & Divide sub menu:

```
---------------------- Multiply & Divide -------------------------
1.  Multiply unary numbers (e.g. 111*11 = 111111)
2.  Divide unary numbers (e.g. 111111/11 = 111)
3.  Back to Main Menu

Selection:
```

By selecting option 1, the user can watch the Turing Machine calculate unary multiplication; by selecting option 2, division. Option 3 will return to the Main Menu.

2.5 Import custom transducer from file

By selecting Main Menu option 4, the user can import a file describing whatever custom machine they like. The machine description file must meet certain conditions, however. These are described by the application at the time of selection:

```
Machine description file must be of the following format:
-----------------------------------------------------------
    => First n lines must describe the transition functions
       of the form:  state, symbol -> state, symbol, moveDir
       where 'state' and 'symbol' are variables and
       'moveDir' is either 'L' or 'R', for move Left or
       move Right, respectively.
    => 'symbol' must be a single character.
    => You must include the commas and '->' delimiter in the
       description.
    => For transitions on "blank", insert empty space between the
       commas and/or delimiters.  (See included .txt files for examples.)
    => No empty lines in the file.
    => After transitions, a line describing the starting state.
    => After starting state, a comma separated list of the
       final states on the machine.  This is a single line.
```

```
Enter the name of the file, including it's relative path from
this working directory:
```

One must strictly adhere to the criterion described in order for the simulation to work as desired. If an incorrect path and/or file name is provided, an error message is displayed:

```
Could not locate file!  Please check path and/or file name and try again.
```

The user is returned to the Main Menu if this error occurs.

2.6 Exit Application

Finally, when the user is ready to exit the application they can select option 5. The application closes and exclaims:

```
Goodbye!
```

3 Application Structure and Code

The application relies on three source files: *main.py*, *machine.py*, and *ui.py*. There is also included a small test-suite in the source file *unitTests.py* which primary tests the *TuringMachine* class in the *machine.py* file. The other two classes, *LanguageAcceptor* and *Transducer* are basically copies of one another that abstract the *TuringMachine* class in ways specific to their respective uses.

3.1 main.py

The *main.py* source file contains a whopping 6 lines of code. It constructs a new *UI* class object (UI stands for "User Interface") and subsequently deletes it upon application exit:

```
import ui

def main():
    Prog = ui.UI()
    del Prog
    return 0

main()
```

3.2 machine.py

This source file contains the three classes which underpin the business-logic of the program, *Turing-Machine*, *LanguageAcceptor*, and *Transducer*. As stated above the *LanguageAcceptor* and *Transducer* classes are approximate copies of one another that merely abstract away the *TuringMachine* class for their respective purposes.

Below is a listing of the public functions included in the three classes. I've excluded the private helper functions and member data for the sake of brevity.

3.2.1 TuringMachine

```
class TuringMachine:

    def getCurrentIndex(self) -> int:
        return self.__TapeIndex

    def getCurrentState(self) -> str:
        return self.__CurrState

    def getCurrentTape(self) -> list:
        return self.__Tape.copy()

    def getTapeLen(self) -> int:
        self.__trimTape()
        return len(self.__Tape)

    def isInFinalState(self) -> bool:
        for i in self.__FinalStates:
            if i == self.__CurrState:
                return True
        return False

    def move(self) -> bool:
        Trans = []
        #Check array boundary conditions
        if (self.__TapeIndex < 0) or (self.__TapeIndex >= len(self.__Tape)):
            Trans = self.__getTransOn('')
        else:
            Trans = self.__getTransOn(self.__Tape[self.__TapeIndex])
        #If returned empty array, halt state
        if len(Trans) == 0:
            return False
        #Else, perform insert/append/replace
        elif self.__TapeIndex < 0:
            self.__Tape.insert(0,Trans[3])
        elif self.__TapeIndex >= len(self.__Tape):
            self.__Tape.append(Trans[3])
        else:
            self.__Tape[self.__TapeIndex] = Trans[3]
        #Move left or right or throw exception
        if Trans[4] == 'L':
            self.__TapeIndex = self.__TapeIndex - 1
        elif Trans[4] == 'R':
            #Special case: if 'deleting' 0 index, don't advance index.
        if Trans[3] == '' and self.__TapeIndex == 0:
            pass
```

```
        else:
            self.__TapeIndex = self.__TapeIndex + 1
        else:
            raise Exception('Machine move definition incorrect or undefined.')
        #Update current state
        self.__CurrState = Trans[2]
        self.__trimTape()
        return True
```

3.2.2 LanguageAcceptor

```
class LanguageAcceptor:

    def printAlpha(self):
        print(self.__Alpha)

    def printFinalStates(self):
        print(self.__FinalStates)

    def printInitialState(self):
        print(self.__InitState)

    def printStates(self):
        print(self.__States)

    def printTransitions(self):
        for p in self.__TransFuncs:
            print(p)

    def run(self, Tape: str, View = True) -> bool:
        self.__TM = TuringMachine(self.__States, self.__Alpha,
            self.__TransFuncs, self.__InitState, self.__FinalStates,
            list(Tape))
        Running = True
        if View:
            print(Tape)
        while Running:
            Plist = self.__TM.getCurrentTape()
            Pstr = ''.join(Plist)
            if View and not DEBUG:
                print('                                        \r' + Pstr,'\r',
                end='')
                time.sleep(0.10)
            Running = self.__TM.move()
            if DEBUG:
                print(self.__TM.getCurrentState(),' ',Pstr)
        if View and not DEBUG:
```

```
            print('\n')
        return self.__TM.isInFinalState()
```

3.2.3 Transducer

```
class Transducer:

    def printAlpha(self):
        print(self.__Alpha)

    def printFinalStates(self):
        print(self.__FinalStates)

    def printInitialState(self):
        print(self.__InitState)

    def printStates(self):
        print(self.__States)

    def printTransitions(self):
        for p in self.__TransFuncs:
            print(p)

    def run(self, Tape: str, View = True) -> bool:
        self.__TM = TuringMachine(self.__States, self.__Alpha,
            self.__TransFuncs, self.__InitState, self.__FinalStates,
            list(Tape))
        Running = True
        if View:
            print(Tape)
        while Running:
            Plist = self.__TM.getCurrentTape()
            Pstr = ''.join(Plist)
            if View and not DEBUG:
                print('                                        \r' + Pstr,'\r',
                    end='')
                time.sleep(0.10)
            Running = self.__TM.move()
            if DEBUG:
                print(self.__TM.getCurrentState(),' ',Pstr)
        if View and not DEBUG:
            print('\n')
        return self.__TM.isInFinalState()
```

If the reader wishes to explore these classes further, I encourage them to explore the source files. There are some global variables that can be used for debugging that are not listed here.

3.3 ui.py

ui.py - standing for "User Interface" - contains all of the menu and sub-menu functionality of the program. After the user makes a menu selection, the construction of the respective classes *LanguageAcceptor* or *Transducer* are made (and subsequently deleted). Most of the exception handling - in the case of a missing machine description file, for instance - is performed within each of the sub-menu modules/functions included in this source file.

Rather than exhaustively list the code for this source file here, the reader is encouraged to view the source file if they wish to see the implementation details. Most of the sub-menu functionality is very similar. Essentially, a *LanguageAcceptor* or *Transducer* class object is constructed, a tape is retrieved from user input, the simulation is run, and the class object is deleted and the user returned to one of the sub-menus.

4 Program Limitations, Bugs, and To-Do's

For all of the a-la-carte functionality in the program, exception handling and testing of the machine description files should ensure that the program will never need to be prematurely aborted. When exceptions are caught, the user is typically prompted and sent back to either the Main Menu or one of the sub-menus. However, there are two possible scenarios wherein the program may throw an uncaught exception, exit unexpectedly, or need to be forceable aborted.

4.1 Custom machine description files.

In the case of custom machine description files (Main Menu option 4, or option number 3 in the Language Acceptors sub-menu), there is the possibility that some combination of symbols in the file may result in unexpected behavior. In the typical case, where the user enters an invalid path and/or file name, the exception is caught, an error prompt is provided, and the program resumes normally.

4.2 Dividing by zero

For the "Divide unary numbers" module, attempting to divide by "zero" will put the Turing Machine into an infinite loop. Attempting to divide by zero occurs when the user enters no characters after the divide ('/') character, like so:

```
Enter the two numbers to add, seperated by a '/' symbol (i.e. 1111/11):  11111/
```

Should the user enter a string like this, they will enter an infinite loop, where the Turing Machine will calculate an ever-growing quotient. If the user were to watch the machine run, it would look something like this:

```
11111/Q111
```

.....

```
11111/Q11111111
```

.....

```
11111/Q111111111111111111111111
```

.....and on and on, forever. This application loop has to be stopped manually be the user using a keyboard interrupt, such as CTRL-C; or through other forceably means.

This bug could likely be corrected with some additions to the machine description file *unaryDivide.txt*.