Assignment 1 - A Turing Machine Simulator

FIT3140 Semester 1, 2015

Individual assignment

Due: Friday 21st March, 11PM
Worth: 5% of final mark.

Alan Turing
Photo Credit UK National Portrait Gallery[1]

Background

If any one person can be identified as the father of computer science, Alan Turing (1912-1954) deserves the title. His contributions to the Allied codebreaking efforts during World War II were the main focus of the recent (excellent, though not entirely historically accurate) movie *The Imitation Game*.

Turing's far-sighted analysis of the question "can a machine think", published in the 1950 paper, "Computing Machinery and Intelligence", remains required reading for anybody vaguely interested in the philosophy of artificial intelligence.

But here we concern ourselves with Turing's first great intellectual contribution - the theoretical construct now known as the Turing Machine.

[1] Image source http://www.npg.org.uk/collections/search/portrait/mw165875/Alan-Mathison-Turing - used under Creative Commons CC-BY-ND http://creativecommons.org/licenses/by-nc-nd/3.0/ s

The Turing Machine

What became known as Turing Machines were introduced to the world in Turing's 1936 paper, "On computable numbers, with an application to the Entscheidungsproblem". Turing's goal was to establish the limitations of *algorithms* - specifically, could an algorithm determine whether a given statements in predicate calculus are true or false. However, to do this, Turing first needed a formal definition of what an algorithm was.

The solution he hit upon was a theoretical computing device which has the following properties:

- An infinitely long "tape" upon which symbols from a limited set, the *alphabet*, could be read and written. One symbol was designated as the "blank" symbol;
- A reader/writer head that can read and one location on the tape at any given moment. The tape can move to the left and right, one step at a time.
- A set of named *states*, which control the behaviour of the machine, of which more will be said later. One of these states is designated the "start state", and zero or more states are designated as "accepting states".
- The tape's starting contents, and the position of the head are defined. By convention, our Turing machines will start at the position of the leftmost defined value on the tape. Any tape position, whose contents were not explicitly defined starts off with the blank symbol. Remember the tape extends infinitely in *both* directions, left and right.

Each *state* has a set of transitions that occur based on the symbol under the head. There is a maximum of *one* transition defined for each symbol, in each state. The transition tells the Turing machine three things to do next when it sees the specified symbol in that particular state:

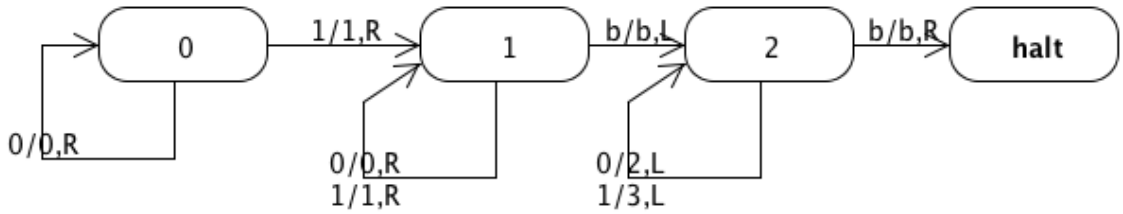
- what symbol from the alphabet to write in the current position to replace the symbol there (note: the replacement symbol may be the same as the current one).
- the new state to enter
- the direction the head should move - left or right.

If *no* transition is defined for a particular symbol, in a particular state, the machine halts. If it is in an "accepting state", it is regarded has having halted with the answer "yes"; if it halts in another state, the machine halts with the answer "no".

Not all Turing machines will halt - some will continue to operate forever. Determining whether this will occur is a major topic of Turing's paper, but is not really of concern here - you don't have to prove the properties of particular Turing machines, just write a program that simulates the behaviour of one.

XML description of a Turing Machine

A Turing machine can be described in a variety of equivalent ways. The states and transitions can be depicted in state transition diagram like this:



This machine has four states, named (imaginitively) 0, 1, 2, and "halt". The names are arbitrary - we call the fourth state "halt" as a descriptive label. We indicate accepting states by writing the name of the state in **bold**.

The transitions are indicated as arrows, with the labels. State 0 has two transitions defined on it - if the machine is in state 0 and sees a 0, it writes a 0 in the same spot (so there's no change), moves right, and stays in state 0. If it sees a 1, it writes a 1, moves right, and changes to state 1.

You will, in the course of the unit, build a graphical interface for creating and running Turing machines, which will appear on the screen something like the above diagram. However, for now, we will use XML files as a convenient way of representing a Turing machine.

The following XML file contains a complete description of the Turing machine depicted above, as well as specifying the alphabet, the starting tape, and the start state.

```
<xml>
<turingmachine>
  <alphabet>0123b</alphabet>
  <initialtape>1010101</initialtape>
  <blank char="b"/>
  <initialstate name="0" />
  <finalstates>
    <finalstate name="halt" />
  </finalstates>
  <states>
    <state name="0">
```

```
      <transition seensym="0" writesym="0" newstate="0" move="R"/>
      <transition seensym="1" writesym="1" newstate="1" move="R" />
      <transition seensym="b" writesym="b" newstate="2" move="L" />
    </state>
    <state name="1">
      <transition seensym="1" writesym="0" newstate="1" move="R"/>
      <transition seensym="0" writesym="1" newstate="1" move="R" />
      <transition seensym="b" writesym="b" newstate="2" move="L" />
    </state>
    <state name="2">
      <transition seensym="0" writesym="2" newstate="2" move="L" />
      <transition seensym="1" writesym="3" newstate="2" move="L" />
      <transition seensym="b" writesym="b" newstate="halt" move="R" />
    </state>
    <state name="halt">
    </state>
  </states>
</turingmachine>
</xml>
```

The file will also be available on Moodle.

Python provides extensive facilities for reading XML files.

I suggest that you use Python's ElementTree library[2] for reading and parsing these files. If you use one of the other Python libraries for XML processing, your demonstrators might not be familiar with them and will be unable to provide as much advice.

Your task

You should write a Python script that defines a function `run_turing(filename)`, which takes one argument, a string representing the name of a file using the format above to specify a Turing machine. You can assume that the file is well-formed - that is, it's legal XML, it defines all the required properties of the Turing machine, and that it is self-consistent (for instance, that the initial contents of the tape contain no symbols not defined in the alphabet).

The script will then print an execution trace of running the defined Turing machine. For each execution step, print the following:

- the number of steps that have been executed so far,

[2] https://docs.python.org/2/library/xml.etree.elementtree.html

- the current state
- the entire tape (you can omit the infinitely long parts of the tape to the left and right that the head had never visited, of course).
- the position of the tape head in relation to the tape. I have done so by printing an asterisk to the left and right of the character which the tape head is "above", but will accept alternative solutions as long is there is **no ambiguity as to where the tape head is**.

At the end you should print with the answer from the execution was "yes" or "no".

I have included a sample program output for the Turing machine above on Moodle. Minor formatting discrepancies between your output and the sample will not be penalized.

You do not have to do anything in particular to handle Turing machines that run forever, or result in a tape so long the computer your simulation executes on runs out of memory.

Your code is intended as a "proof-of-concept-prototype", so your code does not have to be robust against invalid input. However, it should be commented so as to serve as a reference for when you implement a production-quality version.

Incremental development

You should develop your program *incrementally*, getting a intermediate parts working and tested before going on to the next. You will be given partial marks for each of the following:

- Ability to read and parse a Turing machine file, even if you can't run the simulation yet.
- Ability to simulate a Turing machine (using an approach that will work with a Turing machine read from a file) , even if you haven't got the parsing working.

Submitting

If your code is in a single Python source file, you need only submit that (appropriately commented) file. Please call this file `turingmachine.py`. Header comments should include your name, and a brief description of what works, and any known bugs.

If your code is in multiple source files, you should submit a ZIP file containing those - not a rar, gzipped tar, or any other archiving format. The file containing the `runTuring` function should be called turingmachine.py

Compatibility and use of third-party code

Your program should run and will be tested under **Python 2.7**. It should use only the classes

distributed with that.

No other third party code should be used, either through inclusion as a library or by cut-and-paste.

Plagiarism and copying

This assignment is an individual assignment. Students should develop their solutions independently. Fill out the electronic plagiarism statement when you submit.

Marking criteria

Your program will be marked on the following criteria:

- Functionality, including correctness of output on Turing machines other than the one supplied.
- Use of appropriate algorithms and data structures
- Appropriate design (including object-oriented features)
- Readability, including comments

If your program has syntax errors, you will be given a failing mark.

Special consideration

If a student faces exceptional circumstances (serious illness or injury, family emergency etc) that prevent them completing the assignment, they may apply for special consideration according to the policy and procedure on the Faculty website:

http://www.infotech.monash.edu.au/resources/student/equity/special-consideration.html

Other late submissions

Students must contact the lecturer if they do not submit the assignment by the deadline. Penalties will apply.

Assignment forum

There will be an assignment forum on the unit Moodle site. This is the preferred venue for assignment clarification-type questions. You should check the assignment forum regularly, as unit teaching staff responses to questions are "official" and can constitute amendments or additions to the assignment spec. Before asking for a clarification, please look in the assignment forum.

You may of course also email the lecturer, or arrange to see them in person if you prefer.