Running the Turing Machine Simulator

The simulator is not the slickest simulator out there (not at all slick) – but it works for our purposes, accommodating programs of arbitrary complexity, and runs in Ruby and so on multiple platforms in an environment with which we are familiar. You should already have Ruby installed (as from Running Ruby).

It will be clear enough from the example file `suc.rb` how to set up the quadruples.

The initial state is,	:state1	
The final state is,	:finish	
Other states are	:stateX	(for integer X)
Zero is,	:o	(lower case O)
One is,	:l	(lower case L)
Blank is,	:blank	
Right is,	:R	
Left is,	:L	

The output will show in a reasonable way. (But the head position will be messed up unless it displays in a monospace font – you should be fine in the Mac terminal, for SCITE, see Running Ruby. To run a program, you need a command of the sort,

```
ruby program.rb BBlolBlolBlol
```

with two leading instances of 'B' and words separated by 'B' to indicate blanks. The program itself does not initially interpret the 'B' as a blank. But it is set up to convert the 'B' into a blank – and you should ignore any output until the program reaches state1. After that, things run as expected.

From the website, obtain `state.rb` and `turing_machine.rb` (which you will never have to open), along with `suc.rb` and `blank.rb`, and put them all in a turing directory. You should be able to open and run the sample program suc.rb with a command,

```
ruby suc.rb BBlol
```

`blank.rb` is a template to create additional Turing machines.

Hint: Whenever you program a simple machine, create it in a separate file, starting with state1. Then when you need the same function in a more complex machine, you can make a copy, use find/replace to change states from the main program part uniformly from 1,2,3 to 11,12,13 or 101,102,103 or the like (replacing :state with :state1 or :state10), and copy the revision into your larger program --- there is no requirement that your states be sequential, so there is no problem about leaving gaps between state numbers!

!!Be sure to comment liberally, or you will never be able to unscramble what you have done!! Also, "outline-style" indentation of code can help clarify subordination relations.

If you get errors associated with 'require_relative' at the start of 'turing_machine' and programs for the machines you create (including 'suc') you may have an older installation of Ruby; change 'require_relative' to 'require' (including in 'blank') and things should be fine.