Class 31 - Turing Machines

Schedule

To be eligible for automatic Red Belt promotion, you need to complete Project 6 (including a good answer to its bonus Problem 8) by **Friday, 22 April**, and have also completed Project 5 (including Problem 8). Otherwise, you may qualify for the Red Belt exam by completing Problems 1-7 of both Project 5 and Problem 6. If you do this by Friday, 22 April, you will be able to take the Red Belt exam starting Monday, 25 April (and have an opportunity to ensure at least an A- in the course before the end of the semester).

Finite State Machines

A *Finite State Machine* is a very simple model of a machine that has a finite amount of memory (unlike the Turing Machine model which has an infinite amount of memory since the tape is infinitely long).

A Finite State Machine consists of:

1. A finite alphabet of symbols (Σ). There is a finite set of symbols that can be written into squares on the tape.
2. A finite set of states (Q). Some of the states may be distinguished as *accepting states* (for the FSM for a Turing Machine, we typically have a distinguished state called "Halt" where the machine stops if that state is reached). One of the states, $q_0 \in Q$, is designated the *start state*.
3. A set of decision rules (δ: $Q \times \Sigma \rightarrow Q$). Each rule is a triple of the form: (*state*, *symbol*) $\rightarrow$ *nextstate*. If the machine is currently in *state* and the next input symbol is *symbol*, after reading the symbol the state is now in *nextstate*.

A Finite State Machine processes its input in order, starting in the start state, q_0, seeing each input symbol only once and following the state rules. You can think of it as a machine that starts with the input on a finite tape, and process that input from left to right, reading one square at a time. An input string is *accepted* by a finite state machine if the state machine ends in an *accepting state* after processing that input string to the end.

Draw the machine described below and explain what language it recognizes (that is, the set of all strings it accepts):

$Q = \{ A, B \}$
$\Sigma = \{ \mathbf{0}, \mathbf{1} \}$
$q_0 = A$
Accepting states: $\{ B \}$

Transitions:

$(A, \mathbf{0}) \rightarrow A$
$(A, \mathbf{1}) \rightarrow B$
$(B, \mathbf{0}) \rightarrow B$
$(B, \mathbf{1}) \rightarrow A$

Can you make a finite statement machine that recognizes the language of balanced parentheses?

S ::= **(** S **)**
S ::= ϵ

Turing Machine

A *Turing Machine* is an abstract model of a digital computer devised by Alan Turing in 1936.

It consists of:

- An infinitely long tape, divided into squares. (The tape is usually thought of as infinitely long only in one direction, but it is equivalent in power to a tape that is infinitely long in both directions.)
- A finite alphabet of symbols. There is a finite set of symbols that can be written into squares on the tape.
- A tape head that can read the alphabet symbol on a single square of the tape. For each step, the tape head reads the symbol at the current tape position, and can move one square either left or right.
- A *finite state machine* that controls the tape head. Unlike the standard FSM described earlier, each output rule for a Turing Machine's FSM has three outputs: the new state, the symbol to write on the tape (in the current square), and the direction to move (Left, Right, or Halt).

What does this Turing Machine do?

(S, **1**) $\rightarrow$ (S, **0**, R)
(S, **0**) $\rightarrow$ (S, **1**, R)
(S, **#**) $\rightarrow$ **Halt**

Go Outcome Turing Machine. Design a Turing Machine that starts with an input tape that starts with a **#**, is followed by a series of **X** and **O** symbols, followed by a **#** at the end. The output should be **1** if there are more **X** symbols than **O** symbols on the input take, and **0** otherwise (they are equal, or more **O** symbols). For example, if the input tape is **#XOXXOOX#** the output tape should be **#1#**.

