CS3231
Tutorial 9

1. (a) Give a one tape Turing Machine which when given $a^n b$ as input, outputs (on the same tape) $a^n b a^n$. Thus one can copy.

   (b) Give a one tape Turing Machine which when given $a^n b a^m$ as input, outputs (on the same tape) a^{n+m}. Thus one can do addition.

   (c) Give an informal description of a one tape Turing Machine which given $a^n b a^m$ as input, outputs (on the same tape) a^{n*m}. Thus one can do multiplication.

2. For the following, you may give informal description of the Turing Machine detailing how it works rather than give the TM itself. You may use as many tapes as you wish, along with whatever worktape alphabet you need.

   (a) Design a Turing Machine to accept $\{a^n b^n c^n \mid n \geq 0\}$.

   (b) Design a Turing Machine to accept $\{a^{n^3+n^2} \mid n \geq 0\}$.

   (c) Design a Turing Machine to accept $\{a^n \mid n \text{ is a prime }\}$.

3. Suppose we define a multi-head Turing Machine as a one tape Turing Machine which has more than one (but prefixed constant) number of heads. The machine can now work based on what each of its heads read (i.e. next state/symbols written/movement of heads, is based on current state and characters read by each of the heads). Formally define a multi-head Turing Machine. Does it give any more power (for language acceptance/function computation) compared to one tape Turing Machine with only one head? Give reasons for your answer.

   Note: Don't forget to define what happens if two (or more) heads try to write on the same cell.

4. In this question we show that the paradigm of function computation by Turing Machines can be "simulated" in some sense by the paradigm of language acceptance.

   Consider any partial function f from Σ^* to Σ^*. For computing f, machine has an input tape, an output tape and maybe several work tapes. On input x, after doing some computation, machine writes $f(x)$ on the output tape and halts. Note that if f is not defined on input x, then the machine does not halt on input x.

   Graph of the function f is given by the language $L_f = \{x\#y \mid x \in \text{domain}(f) \text{ and } f(x) = y\}$, where we assume $\# \notin \Sigma$.

   Show that some Turing Machine computes the partial function f iff there is another Turing Machine which accepts L_f.

5. Consider a non-deterministic Push Down Automata, which has two push down stacks instead of one. The moves of this machine depends on the input, as well as the content of top of the two stacks. Is 2 stack NPDA as powerful as Turing Machine (for language acceptance)?

6. Consider the following assembly language Z.

(a) There is one input variable X, and one output variable Y, and any fixed number of other temporary variables. Each variable can hold a non-negative integer.

(b) The instructions may or may not have labels.

(c) The instructions in the assembly language are of the form:

   i. $V \leftarrow V + 1$. Meaning: Add one to variable V.

   ii. $V \leftarrow V - 1$. Meaning: Subtract one from variable V. If $V = 0$, then subtraction doesn't have any effect (i.e. V remains 0).

   iii. IF $V \neq 0$, then GOTO L. Meaning: If $V \neq 0$, then go to instruction number L. Otherwise go to the next instruction.

(d) A program is any finite sequence of instructions of above form.

   In execution of any program in the above language, initially all temporary variables and output variable have value 0 (input variable has value as given in input). The execution of program ends (if ever) when there is no next instruction to execute (in case of instructions of type (i) and (ii) above, the next instruction is the instruction immediately after the current instruction in the list of finite instructions). When the program halts, output is the value of the output variable Y.

Does above assembly language have the same power (as far as function computation goes) as a Turing Machine? Give justification for your answer.