Solutions of Cpt S 317 Homework #10

1. Build a Turing machine accepting $(b+c)^+\#a^+$.

M makes sure that reads at least one b or one c, and at least one a. Before the delimiter #, M only reads b or c (no a), and after the delimiter, M only reads a's.

Construction:

$\delta(q_0, b) = (q_1, b, R)$ (reads at least one b)

$\delta(q_0, c) = (q_1, c, R)$ (reads at least one c)

$\delta(q_1, *) = (q_1, *, R)$ with $* = b, c$ (skip all b and c and seek #)

$\delta(q_1, \#) = (q_2, \#, R)$ (starts to read the second block)

$\delta(q_2, a) = (q_3, a, R)$ (reads at least one a)

$\delta(q_3, a) = (q_3, a, R)$ (skip all a)

$\delta(q_3, B) = (q_4, B, S)$ (at the end of the input, accept. I won't get to this if there was a b, c, or # in the second block)

q_4 is the final state.

2. Build a Turing machine accepting $\{x\#x^r : x \in \{a,b\}^+\}$.

Ideas. Naturally, the input is expected in two blocks: one is before the delimiter #, the other is after #. M marks the leftmost unmarked symbol in the first block and compares and marks the rightmost unmarked symbol in the second block. M repeats this process until every symbol is marked.

Construction:

$\delta(q_0, *) = (q_*, \bar{*}, R)$ with $* = a, b$ (make sure to read at least one a or b, and mark and remember it)

$\delta(q_*, *_1) = (q_*, *_1, R)$ with $* = a, b, *_1 = a, b$ (skip all unmarked symbols and seek the delimiter #)

$\delta(q_*, \#) = (q^1_*, \#, R)$ with $* = a, b$ (now I am processing the second block)

$\delta(q^1_*, *_1) = (q^1_*, *_1, R)$ with $* = a, b, *_1 = a, b$ (skip all unmarked symbols and seek the leftmost marked symbol (or a B if initially)

$\delta(q^1_*, \bar{*_1}) = (q^2_*, \bar{*_1}, L)$ with $* = a, b, *_1 = a, b$ (now I am looking at the rightmost unmarked symbol in the second block)

$\delta(q^1_*, B) = (q^2_*, B, L)$ with $* = a, b$ (This is the case for B mentioned above. now I am looking at the rightmost unmarked symbol in the second block)

$\delta(q^2_*, *) = (q^3, \bar{*}, S)$ with $* = a, b$ (compare and mark this unmarked symbol. M must find this unmarked symbol, otherwise, the input is not in L, since the first block is longer than the second block.)

$\delta(q^3, *) = (q^3, *, L)$ with $* = a, b$ (move back to the delimiter)
$\delta(q^3, \#) = (q^4, \#, L)$ (now I am in the first block)
$\delta(q^4, *) = (q^4, *, L)$ with $* = a, b$ (skip all the unmarked symbols and seek the first marked symbol)
$\delta(q^4, \bar{*}) = (q^5, \bar{*}, R)$ (now I am look at the first unmarked symbol in the first block)
$\delta(q^5, *) = (q_*, \bar{*}, R)$ with $* = a, b$ (repeat the process same as from q_0)
$\delta(q^5, \#) = (q^6, \#, R)$ (all the first block are marked, so make sure that all the second block are marked)
$\delta(q^6, \bar{*}) = (q^6, \bar{*}, R)$
$\delta(q^6, B) = (q^7, B, S)$ (accepting if I can hit a B, i.e., all the symbols in the second block are marked)
q^7 is the final state.

3. Turing machines are definitely stronger than PDAs (why?). But it is interesting to investigate some restricted forms of TMs and see if the restrictions we apply could maintain the computing power of TMs. Here is an example. Consider a restricted TM M, called a *one-turn* Turing machine, such that during any executions on any input, M makes at most one turn on the tape (i.e., the tape head either always moves to the right, or moves to the left later but never moves to the right again. However, the tape head could stay at any moment.) Show that languages accepted by one-turn Turing machines are context-free.

It suffices to show that a one-turn TM M can be simulated by a PDA M'. The idea is really simple. We use a pushdown stack to record the content (from the leftmost to the head) of the Turing tape. Whenever M does $\delta(q, a) = (p, b, R)$, M' does
change state from q to p;
replace top of stack a by b;
push the next input symbol into the stack by reading an input symbol.
Whenever M does $\delta(q, a) = (p, b, S)$, M' does
change state from q to p;
replace top of stack a by b;
push the next input symbol into the stack;
do not read any input symbol.
Whenever M does $\delta(q, a) = (p, b, L)$, M' does
change state from q to p;

replace top of stack a by b;
pop the stack top from th stack;
do not read any input symbol.
In fact, M' is a deterministic PDA. The accepting states are the final states of M. M' also makes sure that when entering an accepting state, it halts (no further moves).

4. Many of you asked why we couldn't have two stacks for PDAs? Now, here comes a problem for you. It is natural to add an extra stack to a PDA. Say M is a *two-stack PDA* if M is exactly the same as a PDA but with two stacks. Each instruction (transition) in M is in the form of

$$(p, \gamma_1, \gamma_2) \in \delta(q, a, b_1, b_2)$$

which means that if M is at state q and reading input symbol a with the top of the two stacks being b_1 and b_2 respectively, then this transition brings M to state p, and replaces the tops b_1 and b_2 of the two stacks to γ_1 and γ_2 respectively. Show (describe) that any Turing machine can be simulated by a two-stack PDA.

Again, the idea is really simple. We use the first stack to store the tape content before the tape head, and use the second stack to store the tape content after the tape head. That is, for a configuration $x(q)y$ of a TM M', the configuration of two-stack PDA M is: the content of the first stack is x (with the rightmost symbol being the top), the content of the second stack is y (with the leftmost symbol being the top), and the control state is q.

Initially, the first stack is empty, and the second stack holds the input word (the rightmost being the top).

Whenever M' does $\delta(q, a) = (p, b, R)$, M' does
change state from q to p;
replace the top of the second stack a by b;
pop the top of the second stack and push this top onto the first stack.
Instructions like $\delta(q, a) = (p, b, L)$ and $\delta(q, a) = (p, b, S)$ can be implemented accordingly.

Thus, a TM M' can be simulated by a two-stack PDA M.