CIS 511: Spring 2012
Problem Set 3 Solutions

(1) Exercise 3.8 (b).

Answer:
1. Scan the tape till you read first zero. If you reach end of input, scan again to see if there is any remaining one, if there is any then reject otherwise accept.
2. Cross the zero, scan the tape to find next zero. If you reach end of input reject, otherwise cross that zero and scan back to begining of input.
3. Scan the tape till you find first one. If you find it before you reach end of input, cross it, scan back to begining of input and go back to step 1, otherwise reject.

(2) Formally define the notion of a PDA with 2 stacks by extending the standard PDA definition in a natural way. Prove that 2-stack PDAs are as powerful as Turing Machines. In other words, prove that they recognize the same set of languages that Turing Machines do.

Answer:
Two stack PDA is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where
Q is the set of states
Σ is the input alphabet
τ is the stack alphabet
$\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \times \Gamma_\epsilon \to Q \times \Gamma_\epsilon \times \Gamma_\epsilon$
$q_0 \in Q$ is the start state
$F \subset Q$ is the set of accept state

Transition function δ takes four arguments current state, current input symbol and symbols popped from the two stacks and returns a triplet consisting of next state and symbols pushed on two stacks.

To show that 2-stack PDA is at least as powerfull as turing machine, we show how we can simulate a standard turing machine by using 2-stack PDA. Idea is to use first stack to hold tape content to the left of the head and second stack to hold remaining tape content. Firstly PDA pushes a special symbol \$ on each stack. It then reads entire input pushing it on first stack. Then it reads the content of the first stack and pushes it on the second stack. At this stage PDA has emptied the first stack and second stack contains the input string. To simlulate any write to current cell, PDA pops and then pushes a symbol on the second stack. If fist stack is not empty, to simulate left head movement, PDA pops a symbol from first stack and pushes it on the second stack. To simulate a right move, PDA pops a symbol from second stack and pushes a symbol on the first stack. If second stack now becomes empty, blank symbol is pushed on it.

It is easy to see that 2-stack PDA is not more powerfull than a turing machine. We can, for example, use 3-tape turing machine to simulate 2-stack PDA. First tape contains input while remaining two tapes have contents of the two stacks. Input read is simulated by moving head on the first tape to the right. Stack pushes are simulated by writing symbol to be pushed on the current tape cell of corresponding tape and moving head to the right. STack pops are simulated by moving head of the corresponding tape to left.

(3) Problem 3.12

Answer:
We show how we can use Turing machine with left reset (LRTM) to simulate ordinary turing machine (OTM)
When OTM moves right, LRTM does the same
When OTM moves left, LRTM does the following
1. Mark the current tape symbol.
2. RESET the head position.
3. Move across the tape shifting the content to right by one cell, but do not change position of the mark.
4. RESET the head position
5. Move across the tape till we reach the marked symbol
6. Unmark the current tape symbol

(4) Problem 3.13

Answer:
The TM variant recognizes class of regular languages. We show this by construcing an NFA that simulates the TM variant as follows.
Let $M = \{Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}\}$ be the TM and $N = \{Q', \Sigma, \delta', q'_{start}, q'_{accept}\}$ be the NFA.

Firstly note that the NFA need not concern about symbols on the left of current head position as TM can not come back and read them. We can use NFA state machine to remember state of TM and current tape symbol, which would make up for $|Q \times \Gamma|$ states. We simulate two moves of the TM variant as below
Stay put move is simulated using epsilon transition as follows
If $\delta(q_1, a) = (q_2, b, S)$ then we add transition $\delta'((q_1, a), \epsilon) = (q_2, b)$
Move right is simulated by reading a symbol from the input as follows
If $\delta(q_1, a) = (q_2, b, R)$ then we add transition $\delta'((q_1, a), c) = (q_2, c)\ \forall c \in \Sigma$

Finally we add 3 extra states q'_{start}, q'_{accept} and q'_{reject} with following transitions.
$Q' = Q \times \Gamma \cup \{q'_{start}, q'_{accept}, q'_{reject}\}$
$\delta'(q'_{start}, a) = \{(q_0, a)\}\ \forall a \in \Sigma$
$\delta'(q_{accept}, a) = \{q'_{accept}\}\ \forall a \in \Sigma$
If $\delta(p, a) = (q_{accept}, b, m)$ where $m = R$ or $m = S$ we add $\delta'((p, a), \epsilon) = \{q_{accept}\}$
If $\delta(p, a) = (q_{reject}, b, m)$ where $m = R$ or $m = S$ we add $\delta'((p, a), \epsilon) = \{q_{reject}\}$

(5) Problem 3.15 (c)

Answer:
Let L be any language and M be the decider for L. We define machine N that decides L^* as follows
Machine N on input w:
1. For each way of dividing w in parts such that $w = w_1 w_2 .. w_n$, run M on each of w_i's, if M accepts all the substrings, accept.
2. If all sub-divisions have been tried without accepting, reject.

For correctness it should be clear that, if N at all halts, it outputs the correct answer. We can see that N is a decider by observing that there are only finite number of ways in which any given string can be subdivided and the fact that M itself is a decider.

(6) Exercise 4.6

Answer: Note that $\mathcal{B}$ is in one-to-one correspondence with the power set of $\mathbb{N}$ (if x is an infinite sequence, then let $S_x = \{n \in \mathbb{N} \mid x_n = 1\}$ – then $S_x = S_y$ iff $x = y$). The proof is completed by noting that for any set S, the cardinality of the power set of S is strictly greater than the cardinality of S.

We prove this last fact by diagonalization. Let f be a map from S to its power set. Let $D = \{s \in S \mid s \notin f(s)\}$. Note that D cannot be $f(s)$ for any $s \in S$, since if $s \in D$, then $s \notin f(s) = D$, and if $s \notin D$, then $s \in f(s) = D$. Thus no map f from S to its power set is surjective, so the power set has strictly greater cardinality.

(7) Problem 4.10

Answer: Here is a decision procedure to determine if a PDA is in $INFINITE_{PDA}$: read in the PDA M and convert it into a CFG G (for example, by the procedure in Lemma 2.27). Now, convert it to an equivalent grammar G' in Chomsky Normal Form. Generate the following graph H: each non-terminal is a vertex, and for a rule $A \to BC$, add edges (A, B) and (A, C). Use DFS or BFS from the start state to see if H has a (directed) cycle. If it does, accept. Otherwise, reject.

If $\langle M \rangle \in INFINITE_{PDA}$, then it has a string of length greater than the pumping length of $L(M)$. Following the proof of the pumping lemma, this means there is a non-terminal X which can derive uXv for some strings u, v (at least one of which is non-empty since the grammar is in Chomsky Normal Form). In particular, this means that in H we can find a cycle which includes X.

Suppose there is a cycle in H. Then it is clear that for some non-terminal X, we can find a derivation $X \to^* uXv$ (where one of u or v is non-empty, since G' is in Chomsky Normal Form). Since we found this cycle from S, we can find $S \to^* aXb$. Thus, $S \to^* au^iXv^ib$ for all i and so $L(M)$ is infinite.

(8) Problem 4.14

Answer: Here is the decision procedure to determine if a given grammar G generates all of 1^*: let p be the pumping lemma constant for G (this can be determined algorithmically – see the proof for details). Check for all $0 \leq i \leq p! + p$ whether $1^i \in L(G)$. If for some i in this range $1^i \notin L(G)$, we reject. If for all $0 \leq i \leq p! + p$, $1^i \in L(G)$, then we accept.

It should be clear that if we find i so that $1^i \notin L(G)$, then $1^* \not\subset L(G)$, so rejecting is correct.

On the other hand, suppose for all $0 \leq i \leq p!+p$, we find $1^i \in L(G)$. Then I claim $1^* \subset L(G)$, so accepting is correct. Consider 1^r for some r, which we can assume to be greater than $p! + p$. Then we can write $r = kp! + s$ where $p < s \leq p + p!$. We know that $1^s \in L(G)$ and since $s > p$, there is a pumping length $0 < l \leq p$ so that for all $n \geq 0$, $1^{s+nl} \in L(G)$[1]. Now note that $r = kp! + s$ and l divides $p!$ since $l \leq p$, so $1^{s+kp!} = 1^r \in L(G)$. As this holds for all s, $1^* \subset L(G)$ as claimed.

(9) Problem 4.16

Answer: Let A, B be two DFAs, and let $|A|, |B|$ denote the number of states in A, B respectively. Let M be a decider for EQ_{DFA} by testing the equality of A and B on all strings up to length $|A||B|$. If they are equal on all such strings, accept. Otherwise, reject.

It is clear that M terminates and accepts or rejects on all inputs. So it suffices to show that this length is enough. Consider the product DFA, $C = A \times B$, where the accept states are (q_A, q_B) where exactly one of q_A, q_B is an accept state in its respective automaton. Then asking equality of A and B is equivalent to asking if C accepts nothing. But if C accepts something, it must accept something of length at most $|C| = |A||B|$, so if A and B disagree on any string x (meaning C accepts x), it must disagree on some string of length at most $|A||B|$.

(10) Problem 4.17

Answer: *If:* Suppose C is a language, and D is a decidable language such that $C = \{x \mid \exists y : (\langle x, y\rangle \in D)\}$. Then D has a decider M. We build a machine T to recognize C. It will iterate over all strings y and simulate M to determine if $\langle x, y\rangle \in D$. If M ever accepts, T accepts. Since M is a decider, it terminates in finite time for each y. So if $x \in C$, then there is y so that M accepts $\langle x, y\rangle$ and so T will accept when it finishes simulating M on $\langle x, y\rangle$. Thus C is recognizable.

Only if: Suppose C is recognizable. Then let M be an enumerator for C. Let $D = \{\langle x, y\rangle \mid x \text{ is the } y\text{-th string output by } M\}$. D is decidable, since on input $\langle x, y\rangle$, we can simulate M until it outputs y strings, and then compare x to the last string output, and accept if they are equal and reject otherwise.

[1]More precisely, we can partition $1^s = uvxyz$ where $|vxy| \leq p$ and $|vy| = l > 0$ so that $uv^ixy^iz = 1^{s+(i-1)|vy|} \in L(G)$ for all i