CS 181 Homework 5

Charles Zhang, 305-413-659

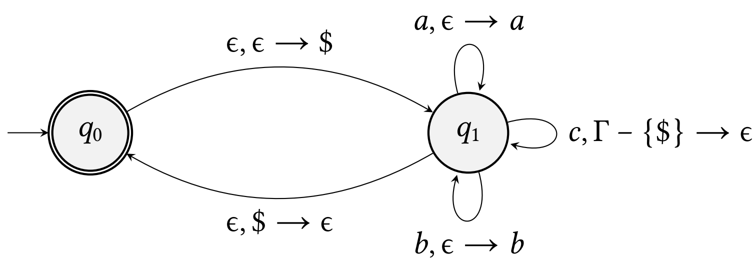
May 2, 2021

Problem 1

$$L_{add} = \{a^i b^j c^k \mid k = i + j\}$$

PDA Model:

$$\Sigma = \{a, b, c\} \quad \Gamma = \{\$, a, b\}$$



Justification:
This PDA starts by pushing \$ onto the stack to indicate an empty stack. It then proceeds to push as and bs onto the stack as they are read from the input string, allowing us to "calculate" $i + j$, as defined by L_{add}. Since the language guarantees that all occurrences of a and b occur before any occurrences of c, we know that after calculating $i + j$, we simply need to check if k is equivalent by counting the remaining cs in the string. We do this by popping a symbol (either a or b) off the stack for each c that is read from the input string. If there are less cs than as and bs, the PDA will remain in state q_1 and reject the input. If there are more as and bs than cs, the PDA will move back into q_0, but won't have reached the end of the input yet, so it will also reject. It will only accept if it moves back into q_0 and there are no more symbols to read, indicating that the number of cs was exactly equal to the number of as and bs.

Problem 2

$$L_2 = \{a^i b^j c^k \mid k = i + j \text{ and } i > j\}$$

Proof (by contradiction):

- Assume L_2 is context-free.
- Let p be the pumping length given by the pumping lemma for CFLs.
- Let s be the string $a^{2p} b^p c^{3p}$, noting that $s \in L_2$, since $i = 2p$, $j = p$, and $k = 3p$, satisfying the condition that $k = i + j$ and $i > j$.
- With s being a member of L_2 and having a length greater than p, the pumping lemma for CFLs guarantees that s can be split into five substrings of the form $s = uvxyz$ such that $^{(1)}$for each $n \geq 0$, $uv^n xy^n z \in L_2$, $^{(2)}|vy| > 0$, and $^{(3)}|vxy| \leq p$.
- Note that the string s can be split into 3 blocks A, B, and C such that A only contains as and is of length $2p$, B only contains bs and is of length p, and C only contains cs and is of length $3p$.
- By condition (3) of the pumping lemma for CFLs, we know that $|vxy|$ has length of at most p.
- This means that vxy can *at most* cross two of the three blocks A, B, and C, as each block has a length of at least p.
- This leaves us with five possible cases to consider: vxy is entirely within block A, vxy is entirely within block B, vxy is entirely within block C, vxy crosses from block A to block B, and vxy crosses from block B to block C.
- Create the string s' by pumping s using $n = t$ such that $t > p$, noting that $t > 0$ by the specification of the language.
- If vxy is entirely within a single block, only the number of one symbol increases; if vxy crosses from A to B, the number of as and bs increases, while the number of cs remains constant.
- In all four of these cases, only a single side of the expression $k = i + j$ increases, while the other remains constant, indicating that the pumped string s' is not a member of L_2.
- This violates the pumping lemma for CFLs, therefore all four of these cases lead to a contradiction.
- The only remaining case is when vxy crosses from B to C.
- By the specification of the case, there must be at least one b in either the substring v or the substring y.
- This means that the number of bs is pumped by t in this case, leading to there being at least $p + p$ bs, since t is defined as some finite value greater than p.
- This violates the condition for L_2 that $i > j$, as the number of bs is now greater than or equal to the number of as.
- In all cases of vxy, the s' is no longer a member of L_2, violating the pumping lemma of CFLs, contradicting the claim that L_2 is a CFL. $\Rightarrow\Leftarrow$

Problem 3

$$A \text{ PS } B = \{w \mid w = a_1 b_1 ... a_k b_k \text{ where } a_i ... a_k \in A \text{ and } b_1 ... b_k \in B \text{ each } a_i, b_i \in \Sigma\}$$

Proof (by construction):

- If the languages A and B are regular, there must exist some DFAs M_A and M_B that accept them.
- To show that FSLs are closed under perfect shuffle, we must construct a finite automaton M based on the machines M_A and M_B that accepts the perfect shuffle of A and B.
- We define $M_A = \{Q_A, \Sigma, \delta_A, q_{0,A}, F_A\}$, $M_B = \{Q_B, \Sigma, \delta_B, q_{0,B}, F_B\}$, and $M = \{Q, \Sigma, \delta, q_0, F\}$.
- We define $Q = (Q_A \times Q_B) \cup \{q_0\}$.
- We define $\delta(q_0, \epsilon) = (q_{0,A}, q_{0,B})$.
- We define $(\delta_A(q_i, a), q_j) \in \delta((q_i, q_j), a)$ for states $q_i \in Q_A$ and $q_j \in Q_B$.
- We define $(q_i, \delta_B(q_j, a)) \in \delta((q_i, q_j), a)$ for states $q_i \in Q_A$ and $q_j \in Q_B$.
- We define $F = (F_A \times F_B)$.

Justification:
The resultant NFA M models the states of M_A and M_B as ordered pairs. By doing this, we can then use the transitions from M_A and M_B, while also taking into account the state of both machines. This means that transitions that M_A would take can now only be taken if M_B is in the appropriate state, and vice versa. We then create a new initial state that uses an ϵ-edge to allow M to model M_A and M_B's initial states. Finally, we create our set of accepting states from the cartesian product of the accepting states of M_A and M_B, as both machines would have needed to end in an accepting state in order for M to accept the input.