Formal Languages Assigment 6

Rachel Hwang

November 21, 2013

1. ***Exercise 0.1*** *Show that the CFL's are closed under the following operations:*

   1. *init, defined as init(L)* $= \{w,\ for\ some\ x,\ wx \in L\}$. *Hint: Start with a CNF grammar for language L.*

      Let $G = (V, T, S, P)$ be a CNF grammar for language L. We can construct a context-free grammar $G' = (V', \{T \cup \epsilon\}, P', S')$ for $init(L)$. The production rules P' consist of the union of P with the following additional rules

      $$\{A' \to BC' \mid A \to BC \in P\}$$
      $$\{A' \to B' \mid A \to BC \in P\}$$
      $$\{A' \to a \mid A \to a \in P, \forall a \in T\}$$
      $$\{A' \to \epsilon \mid A \to a \in P, \forall a \in T\}$$

      And so our start variable will be S', which takes rules of the form $S' \to BC'|B'$ where $S \to BC$ in G. Every sentential form we can derive from S' must either be from $a \in T^*$, or else end with a prime variable, which can be nulled. Thus, $L(G')$ contains every $w \in L$ as well as every y where $ya \in L$, $a \in T^*$.

   2. *The operation L/a, defined as* $L/a = \{w : wa \in L\}$. *For example, if* $L = \{a, aab, baa\}$ *then* $L/a = \{\epsilon, ba\}$. *Hint: Again, start with a CNF grammar for L.*

      Let $G = (V, T, S, P)$ be a CNF grammar for language L. We can construct a context-free grammar $G' = (V, T, P', S')$ for L/a. The set of variables terminals will be the same as in G. The production rules P' consist of the union of P with the following additional rules

      $$\{A' \to BC' \mid \text{where } A \to BC \in P\}$$
      $$\{A' \to \epsilon \mid \text{where } A \to a \in P\}$$

      And so our start variable will be S', which takes rules of the form $S' \to BC'$ where $S \to BC$ in G. Because we start from S', any sentential form we can produce from S' will end with a prime variable. Because for any production rule $A \to a$, (and we want to remove a), A' only produces the empty string, any occurance of a at the end of a string will be removed under our new production rules. However, if a occurs somewhere in the middle of the word, it is preserved because the variable producing it will not be marked as a prime, since by construction primes occur only in the right-most position in a sentential form. a is the only character nulled, so for every string of the form $wa \in L$, $w \in L(G')$.

   3. *cycle, defined as cycle(L)* $= \{w : we\ can\ write\ w\ as\ w = xy\ ,\ such\ that\ yx \in L\}$. *For example, if* $L = \{01, 011\}$, *then cycle(L)* $= \{01, 10, 011, 110, 101\}$. *Hint: Try a PDA-based construction.*

      See attached.

2. ***Exercise 0.2*** *Give the formal proof of the following theorem (Thm 7.25 of the textbook): that the CFL's are closed under reversal.*

   Given CNF grammar $G = \{V, T, P, S\}$ of a CFL, we can generate $G' = \{V, T, P', S\}$, the grammar of the reverse. For every rule $A \to BC \in P$, we instead have $A \to CB \in P'$.

Claim: Every sentential form in G' is the reverse of a sentential form in G.
Proof: By induction on the length of the sentential form.
Basis: For length 1 (or zero), this is trivial, since for a rule $A \to a$, the produced string is its own reverse. Now, assume our claim is true for all strings of length k. For the $k+1$ case, let $\alpha = \beta BC\gamma$ where $\alpha \in V^*$, $S \Rightarrow^* \alpha$, $|\alpha| = k+1$ and production rule $A \to BC \in P'$ for some A (since our grammar is in CNF. So, we know there is some sentential form $\alpha' = \beta A\gamma$. Since $|\alpha'| = k$, by our assumption, it corresponds to some $x' = \beta^R A\gamma^R$. But by the construction of our P', we also know that since $A \to BC \in P'$, $A \to CB \in P$. So we have $\alpha' = \beta A\gamma \leftrightarrow x' = \beta^R A\gamma^R$ and $\alpha' = \beta CB\gamma \leftrightarrow x' = \beta^R CB\gamma^R$. So we have shown the length $=$ $k+1$ case. Thus, by induction, we have proved the claim.

3. ***Exercise 0.3*** *The shuffle of two strings w and x is the set of all strings that one can get by interleaving the positions of w and x in any way. More precisely, shuffle(w,x) is the set of strings z such that*

   (a) *Each position of z can be assigned to w or x, but not both.*

   (b) *The positions of z assigned to w form w when read from left to right.*

   (c) *The positions of z assigned to x form x when read from left to right.*

   For example, if w = 01 and x = 110, then $shuffle(01, 110) = \{01110, 01101, 10110, 10101, 11010, 11001\}$.

   To illustrate the necessary reasoning, the fourth string, 10101, is justfied by assigning the second and fifth positions to 01 and positions one, three, and four to 110. The first string, 01110, has three justifications. Assign the first position and either the second, third, or fourth to 01, and the other three to 110.

   1. ***What is shuffle(00, 111)?***

      shuffle(00, 111) $= \{00111, 01011, 01101, 01110, 10110, 11010, 11100, 10011, 11001, 10101\}$

   2. ***What is shuffle(L1, L2) if L1 =*** $\boldsymbol{L}(0^*)$ ***and L2 =*** $\{0^n1^n | n \geq 0\}$***?***

      shuffle(L1, L2) $= \{0^n(0^*10^*)^n\}$

   3. ***Show that if L1 and L2 are both regular languages, then so is shuffle(L1, L2). Hint: Start with DFA's for L1 and L2.***

      Given a DFA $M_1 = (Q_1, \Sigma_1, \delta_1, q_{01}, F_1)$ for $L1$ and a DFA $M_2 = (Q_2, \Sigma_2, \delta_2, q_{02}, F_2)$ for $L2$, we can construct an NFA which exactly describes shuffle(L1, L2), call it

      $$M = (Q, \Sigma, \delta, (q_{o1}, q_{o2}), F)$$

      Since we must use both alphabets, $\Sigma = \Sigma_1 \cup \Sigma_2$. Each state in M will be an ordered pair of states, (q, p). $Q = \{(q, p) | q \in Q_1, p \in Q_2\}$, and so the start state is (q_{01}, q_{02}). The set of final states $F = \{(q_1, q_2) | q_1 \in F_1, q_2 \in F_2\}$. The transition function is defined as follows

      $$\delta((q_1, q_2), a) = \begin{cases} (\delta_1(q_1), q_2), & \text{if } a \in \Sigma_1. \\ (q_1, \delta_2(q_2)), & \text{if } a \in \Sigma_2. \end{cases}$$

      This accurately describes shuffle(L1, L2) because M allows traversal of both strings at once. Since we keep track of our current state in both machines, only valid transitions in each machine are allowed. However, because our set of final states include only ordered pairs consisting of a final state from M_1 and one from M_2, we ensure that we can only accept a string once we have read both a valid string in L1 and in L2. The ordered pair construction merely allows us to 'interleave' the two strings, since at anytime, we may choose to transition in either M_1 or M_2.

   4. ***Show that if L is a CFL and R is a regular language, then shuffle(L, R) is a CFL. Hint: start with a PDA for L and a DFA for R.***

      Given a PDA (which accepts by empty stack) $M_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, q_{01}, Z_0, F_1)$ for L and a DFA $M_2 = (Q_2, \Sigma_2, \delta_2, q_{02}, F_2)$ for R, we can construct an PDA (accepting by final state) which exactly describes shuffle(L, R), call it

      $$M = (Q, \Sigma, \Gamma, \delta, (q_{o1}, q_{o2}), Z_0, F)$$

Since we must use both alphabets, $\Sigma = \Sigma_1 \cup \Sigma_2$. Each state in M will be an ordered pair of states, (q, p). $Q = \{(q, p) | q \in M_1, p \in M_2\}$, and so the start state is (q_{01}, q_{02}). The start stack symbol is the same as in M_1. The set of final states $F = \{(q_1, q_2) | q_1 \in F_1, q_2 \in F_2\}$. The transition function is defined as follows

$$\delta((q_1, q_2), a, X) = \begin{cases} ((p_1, q_2), \gamma), & \text{if } a \in \Sigma_1,\ \delta_1(q_1, a, X) = (p_1, \gamma). \\ (q_1, \delta_1(q_2), X), & \text{if } a \in \Sigma_2. \end{cases}$$

This accurately describes shuffle(L, R) because M allows traversal of both strings at once. Since we keep track of our current state in both machines, only valid transitions in each machine are allowed. Note that because only M_1 manipulates the stack, all transitions in M_2 do not effect the stack, reading a stack symbol and replacing it with the same. As for acceptance, we accept by final state. Because our set of final states include only ordered pairs consisting of a final state from M_1 and one from M_2, we ensure that we can only accept a string once we have read both a valid string in L and in R. The ordered pair construction merely allows us to 'interleave' the two strings, since at anytime, we may choose to transition in either M_1 or M_2.

5. ***Give a counterexample to show that if L1 and L2 are both CFL's, then shuffle(L1, L2) need not be a CFL.***

   We can pick $L1 = \{a^n b^n\}$ and $L2 = \{a^n b^n\}$. We can use the pumping lemma to show that shuffle(L1,L2) is not a CFL.

   For some number n, let us pick a string in $L, z = a^n b^n c^n d^n$, where $z = uvwxy$, $|vwz| \le n$ and $vx \ne \epsilon$. By the pumping lemma, we can pick any p and $uv^p wx^p y$ will be in L.

   However, since we know $|vwz| \le n$, vwz is either contained in the sustring of one symbol, or it straddle two adjacent symbols. If vwz consits of only one symbol, then vwz has n of three different symbols and fewer than n of the fourth symbol. Thus, it cannot be in L. If vwz straddles two symbols, say the a's and b's, then vwz is missing either some a's or some b's. Suppose it is missing a's. As there are n c's, vwz cannot be in L. In short, no matter how vwz is picked, we cannot get all the letters, and so pumping will result in an uneven increase, more of some, but not all of the letters. This violated the constraint that there must be the same number of each. We have contradicted the assumption that L is a CFL, and conclude that it is not.

4. ***Exercise 0.4*** *Design Turing machines for the following languages:*

   1. *The set of strings with an equal number of 0's and 1's.*

      $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_4\})$

State	0	1	X	Y	B
q_0	(q_1, X, R)	(q_2, X, R)	$-$	$-$	(q_4, B, R)
q_1	$(q_1, 0, R)$	(q_3, Y, L)	(q_1, X, R)	(q_1, Y, R)	$-$
q_2	(q_3, Y, L)	$(q_2, 1, R)$	(q_2, X, R)	(q_2, Y, R)	$-$
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_0, X, R)	(q_3, Y, L)	$-$
q_4	$-$	$-$	$-$	$-$	$-$

      q_0: we first check the leftmost input symbol, and decide next state.
      If we read B, switch to accepting state q_4. If we read 0, replace with X, switch to q_1, and move right. If we read 1, switch to q_2, replace with X, and move right. If we read X or Y, fail.

      q_1: we are checking for a 1 to the right.
      If we read 1, replace with Y, switch to q_3, and move left. If we read a 0, X or Y, leave it and continue right. If we read B, fail.

      q_2: we are checking for a 0 to the right.
      If we read 0, replace with Y, switch to q_3, and move left. If we read a 1, X or Y, leave it and continue right. If we read B, fail.

      q_3: we are moving to leftmost input var.
      If we read X, leave it, switch to q_0, move right. We should not see B. If 0, 1 or Y, leave it, stay in state

q_3, move left.

2. $\{a^n b^n c^n | n \geq 1\}$

$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{a, b, c\}, \{a, b, c, X, Y, Z, B\}, \delta, q_0, B, \{q_4\})$
where δ is

State	a	b	c	X	Y	Z	B
q_0	(q_1, X, R)	$-$	$-$	$-$	$-$	$-$	$-$
q_1	(q_1, a, R)	(q_2, Y, R)	$-$	$-$	(q_1, Y, R)	$-$	$-$
q_2	$-$	(q_2, b, R)	(q_3, Z, R)	$-$	$-$	(q_2, Z, R)	$-$
q_3	(q_3, a, L)	(q_3, b, L)	(q_3, c, L)	(q_3, X, L)	(q_3, Y, L)	(q_3, Z, L)	(q_4, B, R)
q_4	$-$	$-$	$-$	$-$	$-$	$-$	$-$

q_0: we are scanning for an a, checking from left to right.
If we hit b, Y, Z or c, we fail. If we see X, leave it and continue right. If we find a, we replace with our X tape symbol (to indicate it has been used) and switch to q_1, and continue right. If we see B, switch to accepting state q_5.

q_1: we are scanning for b, checking from left to right.
If we hit a blank (B), c, X or Z, we fail. If we find a or Y, we leave it as is, and continue to the right. If we find b, we replace with our Y tape symbol and switch to q_2, and continue right.

q_2: we are scanning for c, checking from left to right.
If we hit a blank (B), a, X or Y, we fail. If we find b or Z, we leave it as is, and continue to the right. If we find c, we replace with our Z tape symbol, switch to q_3, and continue right.

q_3: we are returning to the rightmost non-blank cell, moving left.
If we hit a blank (B), leave it blank, move one right and swtich to state q_0. If we find anything else, leave it as is, and continue to the left.

q_4: Accepting state.

3. $\{ww^R | w \textit{ is any string of 0's and 1's}\}$

$M = (\{q_0, q_1, q_2, q_3, q_4, q_5\}, \{0, 1\}, \{0, 1, X, B\}, \delta, q_0, B, \{q_5\})$

State	0	1	X	B
q_0	(q_1, X, R)	(q_1, X, R)	$-$	(q_5, B, R)
q_1	$(q_1, 0, R)$	$(q_1, 1, R)$	(q_3, X, L)	(q_3, B, L)
q_2	$(q_1, 0, R)$	$(q_1, 1, R)$	(q_4, X, L)	(q_4, B, L)
q_3	(q_5, X, L)	$-$	$-$	$-$
q_4	$-$	(q_5, X, L)	$-$	$-$
q_5	$(q_5, 0, L)$	$(q_5, 1, L)$	(q_0, X, R)	(q_0, B, R)
q_6	$-$	$-$	(q_6, X, R)	(q_7, B, R)
q_7	$-$	$-$	$-$	$-$

q_0: we are checking for the empty string, otherwise consume first symbol.
If we read B, switch to accepting state q_3. If we read 0, replace with X, switch to state q_1, move right. If we read 1, replace with X, switch to state q_2, move right.

q_1: we are looking for the rightmost end of the unconsumed input, in pursuit of finding a 0.
If we read X or B, leave it as is, switch to state q_3, move left. If we read anything else, leave it and continue right.

q_2, we are looking for the rightmost end of the unconsumed input, in pursuit of finding a 1.
If we read X or B, leave it as is, switch to state q_4, move left. If we read anything else, leave it and continue right.

q_3, we are looking for a 0.
If we read 0, mark it X, switch to state q_5 and continue left. If anything else, fail.

q_4, we are looking for a 1.
If we read 1, mark it X, switch to state q_5 and continue left. If anything else, fail.

q_5, we are looking for the leftmost unconsumed input cell.
If we read X or B, leave it as is, switch to state q_0, move right. If anything else, leave it, and continue left.

q_6, we are checking to see if we have only X's (done).
If X, leave it and continue right. If B, switch to accepting state q_7. If anything else, fail.