Solutions: Homework 5
by Chien-Chung Huang

1

1 Answer:

$S_0 \to S_1|S_2|S_3$

$S_1 \to 00S_1|01S_1|11S_1|10S_1|A$

$S_2 \to BC$

$S_3 \to CB$

$B \to ABA|0$

$C \to ABA|1$

$A \to 0|1$

2

2.1 answer $\{x \in \{0,1\}^* : \text{the number of 0s are twice the number 1s}\}$

2.2 answer [**This solution needs fixing. Please ignore it for now. It will be fixed soon, we hope.**]

We use the technique discussed in the class, i.e., drawing a graph whose y-axis increases/decreases with the terminal read. Note that in this case, every "1" read decreases the height by 2 units and every "0" increases by 1 unit.

We will prove by indunction. The basic case are either 100 or 001 or 010, which can respectively be generated by $S \to 00S1 \to 001$, $S \to 1S00 \to 100$, and $S \to 0S1S0 \to 010$. And all these three strings will not change the height of the graph.

Assume that all strings generated by the CFG whose lengths is smaller than m will not change the height.

Now consider string x, whose length $|x|$ is m.

We now enumerate all possible cases:

$x = 0y0$. We claim that string y will bring the graph from the height 0 to 0. Consider the rule $S_0 \to 0S1S0$. S, whose length is smaller than m, will not change the height by induction, 1 will decrease the height by 2. Reading the two 0's in the beginning and in the end will increase the height by 2. Thus the claim is proved.

$x = 1y0$. We consider two possible cases. (1)$x = 1y'00$. In this case, $S_0 \to 1S00$. The length of S is smaller than m and will not change the height by induction. Therefore, $1S00$ will also not change the height. (2) $x = 1y'10$. In this case, $S_0 \to SS \to 1S000S1S0 \to 1S000S10$. The two Ss will not change the height. Thus, $1S000S10$ also will not change the height.

$x = 1y1$. We claim that y will bring the height from -2 to 2. Consider $S_0 \to SS \to 1S0000S1$. For both S whose length is smaller than m, thus we know that $S0000S$ will incrase the height by 4 unit. Hence the result.

$x = 0y1$. This can be proved in a symmetrical way as we have done in $x = 1y0$.

3

3.1 Answer:

It's not a CFL.
let $\{a^n b^n c^m \mid n \leq m \leq 2n\} = L$
Assume it's a CFL, let p be the constant mentioned in PL. Consider the string $w = a^p b^p c^{2p}$.
we can split w into $uvwyz$ satisfying all the three conditions of PL.
if vxy is contained in all a's, when we pump up, the number of a's will be larger than the number of b's. So the result is not in L.
if vxy is contained in all b's, when we pump up, the number of b's will be larger than the number of a's. So the result is not in L.
if vxy is contained in all c's, when we pump up, the number of s's will be larger than two times the number of a's. So the result is not in L.
if vxy is contained in a's and b's, when we pump up p times, the relationship $n \leq m \leq 2n$ will no longer hold. So the result is not in L.
if vxy is contained in b's and c's, when we pump up p times, either the relationship $n \leq m \leq 2n$ will no longer hold, or the number of a's and b's is not equal. So the result is not in L.
it's a contradiction to PL, thus L is not a CFL.

3.2 Answer:

L is not a CFL. For a proof, assume L is a CFL. Let p be the number mentioned in PL. Let $s = a^p b^{p^2}$. Clearly s is in L and $\mid s \mid$ is at least p. By PL, there exist u, v, x, y, z such that $s = uvxyz$, $\mid vxy \mid \leq p, \mid vy \mid > 0$, and $uv^i xy^i z$ is in L for all $i \geq 0$. If vy consists only one type of symbol, uv^2xy^2z is clearly not in L, contradicting PL. if vy consists of two different symbols, suppose there are k a's in vy(k is at least 1), then the maximum number of b's in vy is p-k. considering uv^2xy^2z, the number of a is p+k and the maximum number of b's is $p^2 + p - k$. $(p+k)^2 = p^2 + 2pk + k^2 > p^2 + p - k$, thus the relationship $m = n^2$ no longer hold. Thus, we have a contradiction to PL. We conclude that L is not a CFL.

3.3 Answer:

We observe that any string in the language is in one of the following two forms:

(1) of the form $uvwv^R x$ where:

u is from $((a \cup b)^* \#)^*$

v is from $(a \cup b)^*$

w is from $(\#(a \cup b)^*)^* \#$

x is from $(\#(a \cup b)^*)^*$

(2) of the form uvx where v is a palindrome and u, x are as above.

In the grammar below, the variables have the following meaning:

<Any> generates $(a \cup b)^*$,

U generates all u as above,

W generates all w as above, and

X generates all x as above,

<Any>$\longrightarrow a$ <Any>| b <Any>| ε

$U \longrightarrow$ <Any> $\# U \mid \varepsilon$

$W \longrightarrow \#$ <Any> $W \mid \varepsilon$

$X \longrightarrow \#$ <Any> $X \mid \varepsilon$

S_1 generates desirable strings of the first form:

$S_1 \longrightarrow UZX$

$Z \longrightarrow aZa \mid bZb \mid W$

S_2 generates desirable strings of the second form:

$S_2 \longrightarrow UPX$

$P \longrightarrow aPa \mid bPb \mid a \mid b \mid \varepsilon$

Finally, start symbol S generates s_1 or s_2:

$S \longrightarrow S_1 \mid S_2$

3.4 Answer:

L is not a CFL. For a proof, assume L is a CFL. Let p be the number mentioned in PL. Let $s = 1^p0^p\#1^p0^{p-1}1$. Clearly s is in L and $\mid s \mid$ is at least p. By PL, there exist u, v, x, y, z such that $s = uvxyz$, $\mid vxy \mid \leq p, \mid vy \mid > 0$, and uv^ixy^iz is in L for all $i \geq 0$. The following is a list of all possible cases: (1) vxy lies entirely before the # symbol, (2) vxy lies entirely after the # symbol, (3) vy contains #, and (4) v contains one or more of the trailing 0's of the first number and y contains one or more of the leading 1's of the second number. In all of these case, uv^2xy^2z is clearly not in L, contradicting PL. We conclude that L is not a CFL.

3.5 Answer:

L is not a CFL. Here is an informal justification. Any string in $(a \cup b)^*$ can be split into a sequence of "blocks", where each block is a string from a^*b^*. By definition of L, a string is in L if and only if one of the following three conditions is true: (1) some block has a different number of a's and b's , or (2) some block has a different number of a's than some other block, or (3) the number of blocks is different from the number of a's in the first block. The PDA for L splits initially into three incarnations, call them M_1, M_2, and M_3. M_1 checks for Condition 1, M_2 checks for Condition 2, and M_3 checks for Condition 3. They work as follows.

M_1 nondeterministically picks a block to check. If that block has an unequal number of a's and b's, it enters accept state and eats up the rest of the input while remaining in the accept state.

M_2 nondeterministically chooses one block, stacks up the a's in that block, and then nondeterministically chooses any one later block and checks if the a's in this block do not tally with the a's already stored on the stack. If they don't tally, it enters accept state and eats up the rest of the input while remaining in the accept state.

M_3 stacks up the a's of the first block and pops one "a" for each block that it encounters. To verify that the number of blocks is unequal to the number of a's in the first block, it lets one incarnation enter the accept state whenever the stack is nonempty. When in accept state, the machine does not consume any input.

4

4.1 Answer:

Assume that the complexity of grammar G is c. Now let us follw Theorem 2.6 step by step:

Remove all the ε rules. We know that for each of the removed rule $A \to \varepsilon$, we have to create 2^X more rules, given X is the number of times A appearing in right-hand side of each rule. Thus, the complexity is now $O(2^c)$.

The next step is to remove the unit rules. Note that there are at most $|V|^2$ units rules. Removing one unit rule at most add the same size of new rules to the set. Thus, the complexity grows up to $O(2^C * |V|^2)$.

Finally, replace all the rules whose right-hand size is bigger than 2. Notice that for each rule, we add at most constant number of rules into this set (more exactly, the number of new rules is the length of the right-hand side -1). Thus, the complexity is still $O(2^C * |V|^2)$.

4.2 Answer: Following the proof of Lemma 2.15 from Sipser's book, we consider the rules separately:

- For those rules $A_{pp} \to \varepsilon$, the complexity is $|Q| * 3$.
- For those rules $A_{pq} \to A_{pr}A_{rq}$, the complexity is $|Q|(|Q|-1)(|Q|-2) * 4$
- For thoese rules $A_{pq} \to aA_{rs}b$, the complexity is $|Q||Q-1||\gamma||\Sigma|^2|Q-2||Q-3| * 5$
  Note: $|Q|, |Q-1|, |Q-2|, |Q-3|$ correspond to the all possible choices of p,q,r,s. Also $|\Gamma|$ is there because that any variable $t \in \gamma$ can be the first character to be pop or pushed. $|\Sigma|^2$ is caused by all the possible choices in the two transition functions $\delta(p, a, \varepsilon), \delta(s, b, t), a, b \in \Sigma$. However, in this question, $|\Gamma|$ and $|\Sigma|$ are constants, so we can ignore them when computing the upperbound.

The total complexity is

$$|Q|(|Q|-1)(|Q|-2)*4+|Q|(|Q|-1)(|Q|-2)*4+|Q||Q-1||\gamma||\Sigma|^2|Q-2||Q-3|*5 = O(|Q|^4)$$

4.3 Answer:

A natural choice for the set of variables is the n states in the $DFA(M) = \{Q, \Sigma, \delta, q_0, F\}$.

We can construct the CFG by the following rules.

- For each state $q_i \in Q$, we create a varible A_{q_i}.
- For every transition function $\delta(q_i, a) = q_j$, we create the rule $A_{q_i} \to aA_{q_j}$.
- For those final states $q_i \in F$, we add the rules $A_{q_i} \to \varepsilon$.

The complexity of our CFG is $|Q||\Sigma| * 4 + |F| * 3 = O(|Q| + |F|) = O(|Q|) = O(n)$.

$|\Sigma|$ can be ignored as the question assumes a constant-sized alphabet.

5

5.1 Answer:

First derivation

⟨ STMT ⟩ →

⟨ IF-THEN ⟩ →

if condition then ⟨ STMT ⟩ →

if condition then ⟨ IF-THEN ⟩ →

if condition then if condition then ⟨ STMT ⟩ else ⟨ STMT ⟩ →

if condition then if condition then ⟨ ASSIGN ⟩ else ⟨ ASSIGN ⟩ →

if condition then if condition then $a := 1$ else $a := 1$.

Second derivation

⟨ STMT ⟩ →

⟨ IF-THEN-ELSE ⟩ →

if condition then ⟨ STMT ⟩ else ⟨ STMT ⟩ →

if condition then ⟨ IF-THEN ⟩ else ⟨ STMT ⟩ →

if condition then if condition then ⟨ STMT ⟩ else ⟨ STMT ⟩ →

if condition then if condition then ⟨ ASSIGN ⟩ else ⟨ ASSIGN ⟩ →

if condition then if condition then a:=1 else a:=1.

5.2 Answer:

⟨ STMT1 ⟩ → ⟨ ASSIGN ⟩| ⟨ IF-THEN ⟩| ⟨ IF-THEN-ELSE ⟩| ⟨ BEGIN-END1 ⟩

⟨ STMT2 ⟩ → ⟨ ASSIGN ⟩| ⟨ IF-THEN-ELSE ⟩| ⟨ BEGIN-END2 ⟩

⟨ IF-THEN ⟩ → if condition then ⟨ STMT1 ⟩

⟨ IF-THEN-ELSE ⟩ → if condition then ⟨ STMT2 ⟩ else ⟨ STMT1 ⟩

⟨ BEGIN-END1 ⟩ → begin ⟨ STMT-LIST1 ⟩ end

⟨ BEGIN-END2 ⟩ → begin ⟨ STMT-LIST2 ⟩ end

⟨ STMT-LIST1 ⟩ → ⟨ STMT-LIST1 ⟩ ⟨ STMT1 ⟩| ⟨ STMT1 ⟩

⟨ STMT-LIST2 ⟩ → ⟨ STMT-LIST2 ⟩ ⟨ STMT2 ⟩| ⟨ STMT2 ⟩

⟨ ASSIGN ⟩ → a:=1