Homework 5
CS 770: Formal Language Theory

Assigned: March 10, 2016 Due on: March 24, 2016

Instructions: This homework has 2 required problems and 1 extra credits problem that can be solved individualy. Please follow the homework guidelines given on the class website. Solutions not following these guidelines will not be graded.

Recommended Reading: Lectures 13 and 14 (Context Free Grammars and Pushdown Automata).

Problem 1. [Category: Design+Proof] Let L be the language consisting all strings over $\{a, b\}$ that have as many as as bs. For example, $abab \in L$ and $\epsilon \in L$ but $a \notin L$.

1. Design a context-free grammar for L. **[5 points]**
2. Prove that your grammar is correct. **[5 points]**
3. Design a PDA to recognize L. You need not prove that your construction is correct, but you should clearly explain the intuition behind your construction. **[5 points]**

Solution:

1. We can inductively define the set L of all strings that have as many as as as bs as follows.
   - $\epsilon \in L$
   - If $w_1 \in L$, $w_2 \in L$ then $w_1w_2 \in L$
   - If $w \in L$ then $awb \in L$
   - If $w \in L$ then $bwa \in L$

   Based on this above inductive definition, we can define a CFG for L as follows. Let $G = (\{S\}, \{a, b\}, R, S)$ where
   $$R = \{S \to \epsilon \mid SS \mid aSb \mid bSa\}$$

2. We first show that for $w \in \{a, b\}^*$ if $S \overset{*}{\Rightarrow}_{\text{lm}} w$ then $w \in L$ by induction on the number of steps in the leftmost derivation of w (see lecture 13 (CFG) for a definition of leftmost derivation).
   - **Base Case:** Suppose $S \overset{*}{\Rightarrow}_{\text{lm}} w$ in one step, then $w \in L$. Observe that if $S \overset{*}{\Rightarrow}_{\text{lm}} w$, then $w = \epsilon$, and ϵ has same number of as as bs. Thus, $\epsilon \in L$.
   - **Ind. Hyp.:** Assume that for any $w \in \{a, b\}^*$, if $S \overset{*}{\Rightarrow}_{\text{lm}} w$ in $< k$ steps (where $k > 0$) then $w \in L$.
   - **Ind. Step:** Suppose $S \overset{*}{\Rightarrow}_{\text{lm}} w$ in k steps. Depending on which rule is used in the first step of derivation, we have different cases to consider.
     - Suppose the derivation is of the form $S \Rightarrow SS \overset{*}{\Rightarrow}_{\text{lm}} w$. Then because we are considering leftmost derivations, the derivation should be of the form $S \Rightarrow SS \overset{*}{\Rightarrow}_{\text{lm}} w_1S \overset{*}{\Rightarrow}_{\text{lm}} w_1w_2$. Thus, $S \overset{*}{\Rightarrow}_{\text{lm}} w_1$ and $S \overset{*}{\Rightarrow}_{\text{lm}} w_2$ in at most k steps. By induction hypothesis, $w_1, w_2 \in L$, and hence $w = w_1w_2 \in L$.

- Suppose the derivation is of the form $S \Rightarrow aSb \overset{*}{\Rightarrow}_{\text{lm}} aub = w$. Then $S \overset{*}{\Rightarrow}_{\text{lm}} u$ in $k-1$ steps and so by induction hypothesis $u \in L$. Thus, $w = aub$ has as many as as bs and so $w \in L$.
- Suppose the derivation is of the form $S \Rightarrow bSa \overset{*}{\Rightarrow}_{\text{lm}} bua$. Again, we have $S \overset{*}{\Rightarrow}_{\text{lm}} u$ in $k-1$ steps, and so by induction hypothesis, $u \in L$. Thus, $w = bua$ has same number of as as bs.

We now prove that if $w \in L$ then $S \overset{*}{\Rightarrow} w$. This is the more difficult direction, and the proof will be carried out by induction on $|w|$.

- **Base Case:** When $|w| = 0$, then $w = \epsilon$. Observe that $\epsilon \in L$. Also since we have a rule $S \to \epsilon$, $S \Rightarrow \epsilon$. Thus, we have established the base case.
- **Ind. Hyp.:** Assume that for any $w \in \{a, b\}^*$ such that $|w| < k$ (for $k > 0$), if $w \in L$ then $S \overset{*}{\Rightarrow} w$.
- **Ind. Step:** Let $w = x_1x_2 \cdots x_k \in L$ be a string of length k; each $x_i \in \{a, b\}$. For a prefix of w of length i (i.e., string $x_1x_2 \cdots x_i$), let d_i be the difference between the number of as in $x_1x_2 \cdots x_i$ and the number of bs in $x_1x_2 \cdots x_i$. That is,

$$d_i = \#_a(x_1x_2 \cdots x_i) - \#_b(x_1x_2 \cdots x_i)$$

Observe that $d_0 = 0$ (since there are 0 as and 0 bs in ϵ), and $d_k = 0$ (since $w \in L$). There are two possible cases: either there is some i between 0 and k (i.e., $0 < i < k$) such that $d_i = 0$, or there is not such i (i.e., for every $0 < i < k$, $d_i \neq 0$). We consider each of these cases.

  - Case for some $0 < i < k$, $d_i = 0$. Then let $u = x_1 \cdots x_i$ and $v \in x_{i+1} \cdots x_k$. Observe that because $d_i = 0$, it follows that $u \in L$ and $v \in L$. Since $|u| < |w|$ and $|v| < |w|$, by the induction hypothesis, $S \overset{*}{\Rightarrow} u$ and $S \overset{*}{\Rightarrow} v$. Thus, we have a derivation $S \Rightarrow SS \overset{*}{\Rightarrow} uS \overset{*}{\Rightarrow} uv = w$.
  - Case for all $0 < i < k$, $d_i \neq 0$. There are different cases to consider based on what the first symbol is.
    Suppose $x_1 = a$. That means $d_1 = 1$. Since for all $0 < i < k$, $d_i \neq 0$, and d decreases by 1 for every b, it must be the case that $d_i > 0$ for all $0 < i < k$. Because if for some $0 < i < k$, $d_i < 0$, then it must be a position $1 < j < i$ such that $d_j = 0$ becasue $d_1 = 1 > 0$. Then in this case, x_k must be b as otherwise d_k would not be 0. Thus, $w = aub$, where $u \in L$. By induction hypothesis, $S \overset{*}{\Rightarrow} u$. Hence, we have a derivation $S \Rightarrow aSb \overset{*}{\Rightarrow} aub = w$.
    When $x_1 = b$, we have $d_1 = -1$. Also, since for all $0 < i < k$, $d_i \neq 0$, and d increases by 1 for every a, it must be the case that $d_i < 0$ for all $0 < i < k$. In addition, we must have $x_k = a$. Thus, $w = bua$ for some $u \in L$. Again by induction hypothesis, $S \overset{*}{\Rightarrow} u$, and so $S \Rightarrow bSa \overset{*}{\Rightarrow} bua = w$.

3. We can design the PDA for this CFG following the idea of parentheses in the lecture. If the string is empty then we accept it, if not, then we push whatever first symbol we see into the stack, and then pop such symbol when we see another kind of symbol. That means if we see a 'a' first, then we keep pushing as until we see bs, when there is no as to pop, i.e. we see a \$ sign, but we still have input and read a b, we change to another state(q_2) then push this b and then start to push bs and pops as when read them(q_5). The case for we see a 'b' is similar it's just the other way round. We accept the string if the stack is empty at the end, reject the string otherwise. The PDA for this CFG is as Figure 1.

■

Problem 2. [Category: Design+proof] Give a context-free grammar that generates the language $A = \{a^ib^jc^k | i = j \text{ or } j = k \text{ where } i, j, k \geq 0\}$. Is your grammar ambiguous? Why or Why not? **[10 points]**

Solution: We can use the idea in Problem 1 to define language A as follows. From Problem 1 we know what's the CFG looks like with equal number of as and bs. Similarly we can easily develop the grammar for strings with equal number of bs and cs. And for this problem, we also need to pay attention to the order of as, bs, and cs in addition. Therefore, we can define the CFG for A as $G = (\{S_a, S_{ab}, S_c, S_{bc}\}, \{a, b\}, R, S)$ where the rules will be like below:

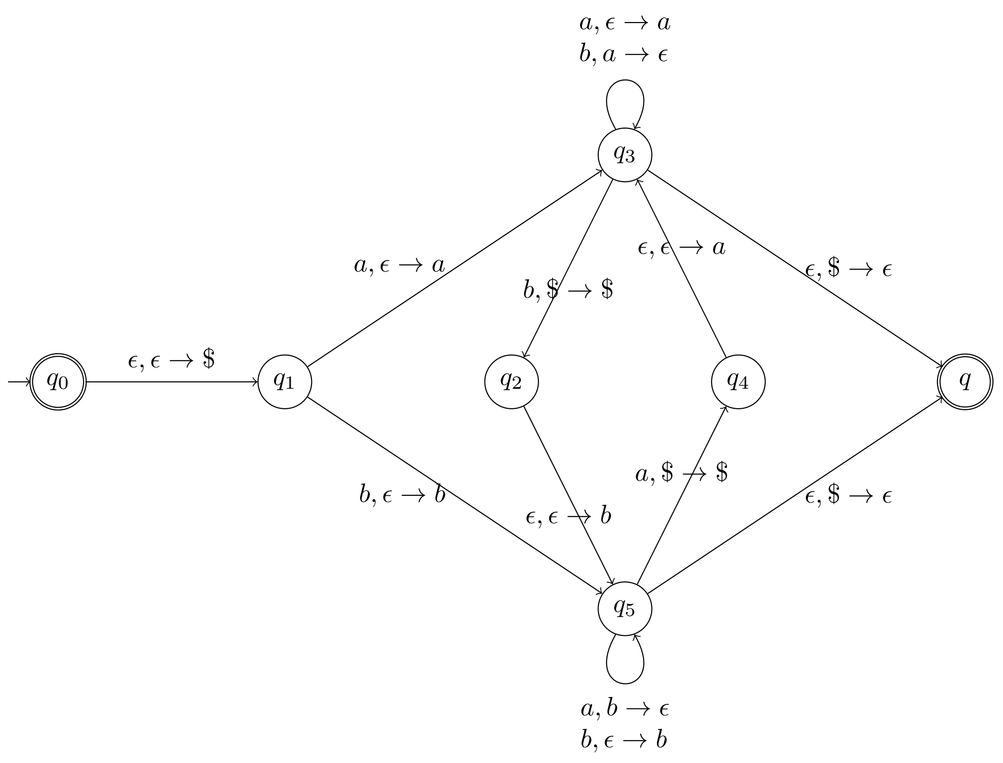

Figure 1: PDA for Problem 1

$$S \to S_{ab}S_c \mid S_aS_{bc}$$
$$S_a \to \epsilon \mid aS_a$$
$$S_c \to \epsilon \mid cS_c$$
$$S_{ab} \to \epsilon \mid aS_{ab}b$$
$$S_{bc} \to \epsilon \mid bS_{bc}c$$

This gramma is ambiguous. To show that let's see the two left derivations for string $w = aabbcc$. We can derive w either as follows:
$S \Rightarrow S_{ab}S_c \Rightarrow aS_{ab}bS_c \Rightarrow aaS_{ab}bbS_c \Rightarrow aabbS_c \Rightarrow aabbcS_c \Rightarrow aabbccS_c \Rightarrow aabbcc$

Or we can derive w as follows:
$S \Rightarrow S_aS_{bc} \Rightarrow aS_aS_{bc} \Rightarrow aaS_aS_{bc} \Rightarrow aaS_{bc} \Rightarrow aabS_{bc}c \Rightarrow aabbS_{bc}cc \Rightarrow aabbcc.$

■

Extra Credits
Problem 3. [Category: Design] Design a PDA to recognize the language $C = \{x\#y \mid x, y \in \{0,1\}^* \text{ and } x \neq y\}$; thus, $C \subseteq \{0, 1, \#\}^*$. You need not prove that your construction is correct, but you should clearly explain the intuition behind your construction. **[10 points]**

Solution: Observe that two strings $x, y \in \{0,1\}^*$ are not equal iff either (a) $|x| \neq |y|$, or (b) there is some i such that $x_i \neq y_i$, where x_i is the ith symbol of x and y_i is the ith symbol of y. On an input string $x\#y$, our PDA will nondeterministically guess whether $x \neq y$ because of condition (a) or because of condition (b).

The PDA will check if its guess was correct. We will now describe how the PDA can check (a) and (b).

For condition (a), the PDA will count the number of symbols in x by pushing something onto the stack for each symbol read. After it reads # it will pop symbols, as it reads y to check how the length of y compares with that of x. If after reading all of y the stack is not empty, then $|x| > |y|$, and if after reading some of y the stack becomes empty then $|y| > |x|$. It will accept in either of these cases.

To check condition (b), the PDA will "guess" the position i and push symbols as it reads from x until it reaches the position it thinks does not match with y. It will read the symbol x_i and store it in its state. It will then ignore the rest of x, and once it reads #, it will start popping the stack as it reads symbols of y, so that it finds the same position i in y. When it reads y_i it will compare this with x_i that it remembered in its state, and if they are not equal, it will accept, else it will reject.

This algorithm can be implemented as a PDA that is shown in Figure 2. The stack alphabet is $\{\$, A\}$. The PDA can be understood as follows. From the initial state, we push a bottom of stack symbol \$ and move to state q. In state q, the PDA nondeterministically decides to either check for condition (a) (in which case it moves through states A, B, C, D) or check for condition (b) (in which case it moves through states $1, 2a, 3a, 4a, 2b, 3b, 4b$). We count the length of x by pushing symbols in state A, and move to state B of reading #. We compare the length of y with that x in B. If we go to state C it means that $|y| < |x|$ and if we go to state D then $|y| > |x|$. Checking for condition (b) proceeds as follows. The PDA nondeterministically decides if it has encountered the position i to check, and if so it moves to either state $2a$, if an 0 is read, or to state $2b$, if a 1 is read; it counts the position i by pushing symbols A onto the state as it reads symbols from x. The computation from $2a$ and $2b$ are similar and we describe only one of them. In state $2a$, we ignore the rest of the input until we read the # in which case we move to state $3a$ to check the ith symbol of y. In state $3a$ we count the symbols (by popping from the stack), and when the stack is empty (i.e., \$ is on the top) then the PDA knows that this is the ith symbol. It checks that this symbol is 1, and if so, it ignores the rest of the input and accepts.

■

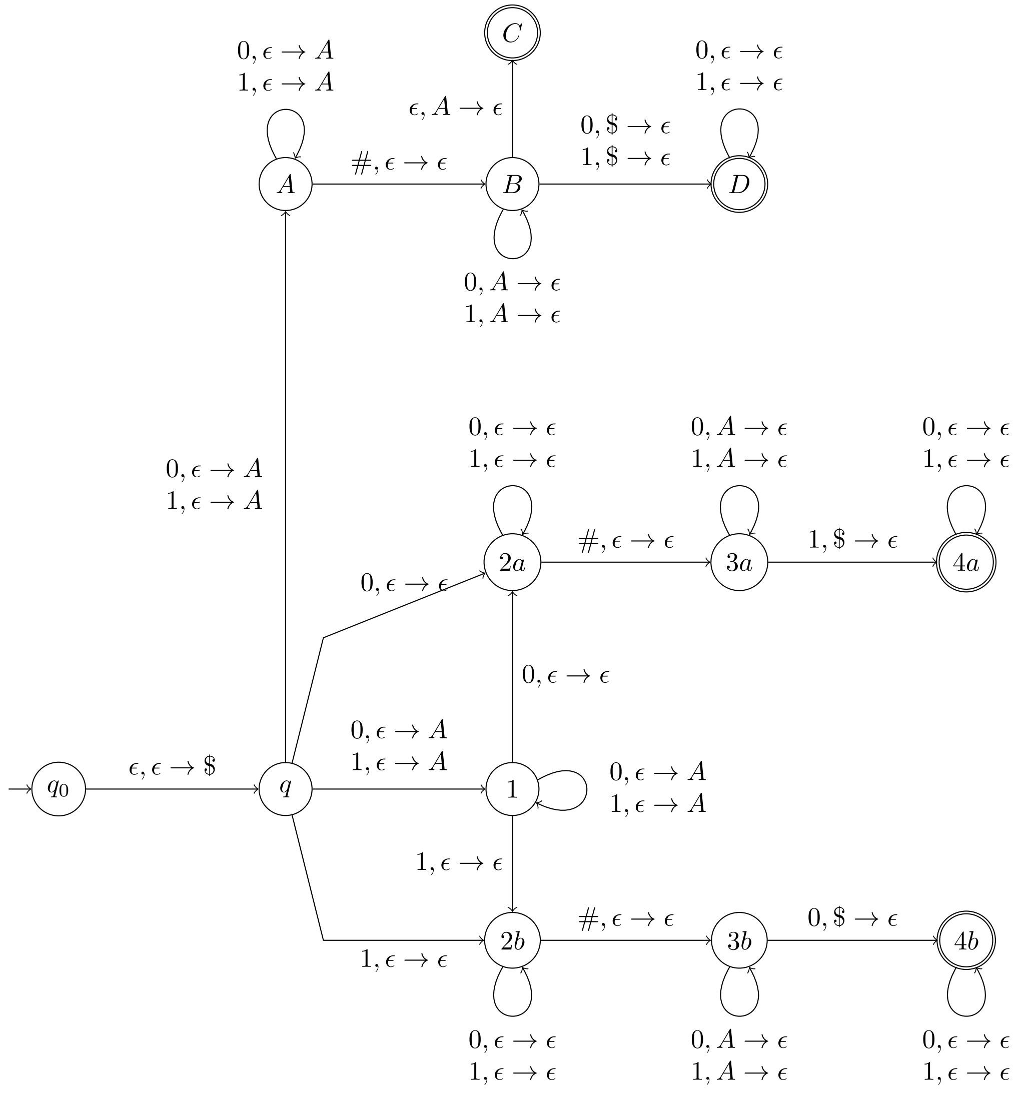
C
$0, \epsilon \to A$
$1, \epsilon \to A$
$\epsilon, A \to \epsilon$
$0, \epsilon \to \epsilon$
$1, \epsilon \to \epsilon$
$0, \$ \to \epsilon$
$1, \$ \to \epsilon$
A
$\#, \epsilon \to \epsilon$
B
D
$0, A \to \epsilon$
$1, A \to \epsilon$
$0, \epsilon \to \epsilon$
$1, \epsilon \to \epsilon$
$0, A \to \epsilon$
$1, A \to \epsilon$
$0, \epsilon \to \epsilon$
$1, \epsilon \to \epsilon$
$0, \epsilon \to A$
$1, \epsilon \to A$
$\#, \epsilon \to \epsilon$
$1, \$ \to \epsilon$
$2a$
$3a$
$4a$
$0, \epsilon \to \epsilon$
$0, \epsilon \to \epsilon$
$0, \epsilon \to A$
$1, \epsilon \to A$
$\epsilon, \epsilon \to \$$
q_0
q
1
$0, \epsilon \to A$
$1, \epsilon \to A$
$1, \epsilon \to \epsilon$
$\#, \epsilon \to \epsilon$
$0, \$ \to \epsilon$
$2b$
$3b$
$4b$
$1, \epsilon \to \epsilon$
$0, \epsilon \to \epsilon$
$1, \epsilon \to \epsilon$
$0, A \to \epsilon$
$1, A \to \epsilon$
$0, \epsilon \to \epsilon$
$1, \epsilon \to \epsilon$

Figure 2: PDA for Problem 3