Name:

1 a	15	
b	15	
c	15	
d	15	
2	15	
3 a	8	
b	8	
c	8	
4 a	5	
b	15	
5 a	8	
b	8	
6	20	
Total	155	

CS 341 Second Midterm Exam Practice

Use extra paper to determine your solutions then neatly transcribe them onto these sheets.

You may use any claim we proved in class as a theorem. But make sure that you only use such a claim when it is exactly what you need. If we have proved something "close" in class, then you must do a complete proof here, but you can use the proof we did in class as a model for your proof.

(1) For each of the following languages L, state whether it is regular, context-free but not regular, or neither. Prove your answer. Make sure, if you say that a language is context free, that you show that it is not also regular.

(a) $\{w \in \{0, 1\}^* : \exists k \geq 0$ and w is a binary encoding (leading zeros allowed) of $2^k+1\}$.

Regular. $L = 0^* (10 \cup 10^*1)$.

(b) $\{a^*b^*c^* - \{a^nb^nc^n : n \geq 0\}\}$.

Context-free not regular. $L = \{a^ib^jc^k : i, j, k \geq 0$ and $i \neq j$ or $j \neq k\}$. L can be accepted by the PDA shown in Example 12.8 except that the top branch should not be present.

If L were regular, then $\neg L$ would also be regular.

$\neg L = \{w \in \{a, b, c\}^*$ with letters out of order$\} \cup \{a^nb^nc^n : n \geq 0\}$. If $\neg L$ is regular, then so is:
$L_1 = \neg L \cap a^*b^*c^*$. But:
$L_1 = \{a^nb^nc^n : n \geq 0\}$, which is not even context-free, much less regular.

(c) $\{(\texttt{ab})^n\texttt{a}^n\texttt{b}^n : n > 0\}$.

Not context-free. We prove this with the Pumping Theorem. Let $w = (\texttt{ab})^k\texttt{a}^k\texttt{b}^k$. Divide w into three regions: the ab region, the a region, and the b region. If either v or y crosses the boundary between regions 2 and 3 then pump in once. The resulting string will have characters out of order. We consider the remaining alternatives:
(1, 1) If $|vy|$ is odd, pump in once and the resulting string will have characters out of order. If it is even, pump in once. The number of ab's will no longer match the number of a's in region 2 or b's in region 3.
(2, 2) Pump in once. More a's in region 2 than b's in region 3.
(3, 3) Pump in once. More b's in region 3 than a's in region 2.
v or y crosses the boundary between 1 and 2: Pump in once. Even if v and y are arranged such that the characters are not out of order, there will be more ab pairs than there are b's in region 3.
(1, 3) $|vxy|$ must be less than or equal to k.

(d) $\{x \in \{a, b\}^* : |x|$ is even and the first half of x has one more a than does the second half$\}$.

Not context-free. In a nutshell, we can use the stack either to count the a's or to find the middle, but not both.

If L were context-free, then $L' = L \cap a^*b^*a^*b^*$ would also be context-free. But it is not, which we show using the Pumping Theorem. Let $w = a^{k+1}b^ka^kb^{k+1}$. Divide w into four regions (a's, then b's, then a's, then b's). If either v or y crosses a region boundary, pump in. The resulting string is not in L because the characters will be out of order. If $|vy|$ is not even, pump in once. The resulting string will not be in L because it will have odd length. Now we consider all the cases in which neither v nor y crosses regions and $|vy|$ is even:
(1, 1): pump in once. The boundary between the first half and the second half shifts to the left. But, since $|vxy| \leq k$, only b's flow into the second half. So we added a's to the first half but not the second.
(2, 2): pump out. Since $|vy|$ is even, we pump out at least 2 b's so at least one a migrates from the second half to the first.
(3, 3): pump out. This decreases the number of a's in the second half. Only b's migrate in from the first half.
(4, 4): pump in once. The boundary between the first half and the second half shifts to the right, causing a's to flow from the second half into the first half.
(1, 2): pump in once. The boundary shifts to the left, but only b's flow to the second half. So a's were added in the first half but not the second.
(2, 3): pump out. If $|v| < |y|$ then the boundary shifts to the left. But only b's flow. So the number of a's in the second half is decreased but not the number of a's in the first half. If $|y| < |v|$ then the boundary shifts to the right. a's are pumped out of the second half and some of them flow into the first half. So the number of a's in the first half increases and the number of a's in the second half decreases. If $|v| = |y|$, then a's are pumped out of the second half but not the first.
(3, 4): pump out. That means pumping a's out of the second half and only b's flow in from the first half.
(1, 3), (1, 4), (2, 4): $|vxy|$ must be less than or equal to k.

(2) Give a decision procedure to answer the question, "Given a context-free grammar G, does G generate at least three strings?"

To construct a decision procedure for this problem, we exploit the same idea that we used as the basis of the Pumping Theorem. Let b be the branching factor of G and let n be the number of G's nonterminals. Then the longest string that G can generate and assign a parse tree with no duplicated nonterminals on any path has length no more than b^n. If G can generate even one string of length greater than that, then that string can be pumped. So G generates an infinite number of strings and thus at least three. Further, if there is at least one such long string, there must also be a shorter one (the result of pumping out from it). We also know that $|vxy| \leq b^{n+1}$. So if G generates any string of length greater than $b^n + b^{n+1}$, then it must also generate at least one whose length is between b^n and b^{n+1} (because we can keep pumping out until we get such a shorter string). So the following algorithm decides the question:

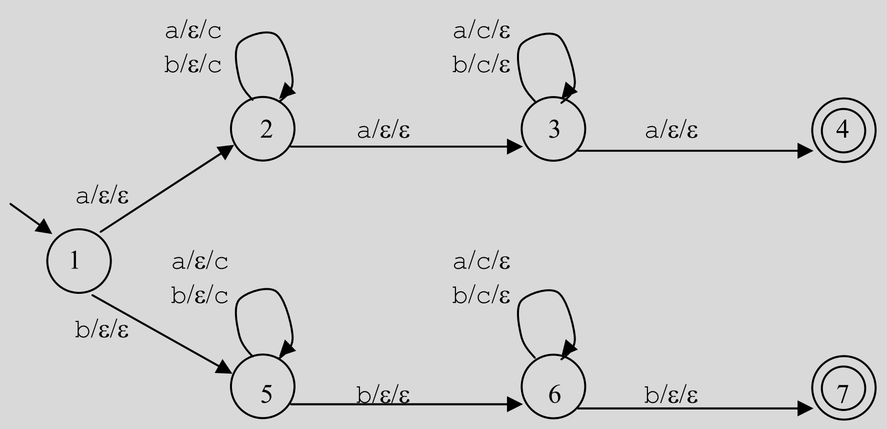
```
1. Examine $G$ and determine $b$ and $n$.
2. Set *count* to 0.
3. For every string $w$ in $\Sigma_G$* such that $|w| \leq b^n$ do:
   3.1. If *decideCFL*($G$, $w$) then *count* = *count* + 1.
4. If *count* = 0 then return *False*.
5. If *count* $\geq$ 3 then return *True*.
6. For every string $w$ in $\Sigma_G$* such that $b^n < |w| \leq b^n + b^{n+1}$ do:
   6.1. If *decideCFL*($G$, $w$) then return *True*.
7. Return *False*.
```

(3) Let $L = \{w \in \{\mathtt{a}, \mathtt{b}\}^*$: the first, middle, and last characters of w are identical$\}$.

(a) Show a context-free grammar that generates L.

$S \rightarrow A \mid B$
$A \rightarrow \mathtt{a}\ M_A\ \mathtt{a}$
$M_A \rightarrow CM_AC \mid \mathtt{a}$
$B \rightarrow \mathtt{b}\ M_B\ \mathtt{b}$
$M_B \rightarrow CM_BC \mid \mathtt{b}$
$C \rightarrow \mathtt{a} \mid \mathtt{b}$

(b) Show a natural PDA that accepts L.



(c) Prove that L is not regular.

If L were regular, then $L' = L \cap$ `ab*ab*a` would also be regular. But we show that it is not by pumping.

Let $w = \mathtt{a}\ \mathtt{b}^k\ \mathtt{a}\ \mathtt{b}^k\ \mathtt{a}$
1 | 2 |3| 4 | 5

If y contains the initial a, pump out. The resulting string will violate the form constraint and thus not be in L. The only other possibility is that y falls completely in region 2 (since it must occur in the first k characters). Note that every string in L has odd length, so y must have even length or we can pump in once and the resulting string will not be in L. Thus y is two or more b's. Pump in once. The middle of the resulting string will fall in the initial b region. Thus it differs from the first and last characters. So the pumped string is not in L.

(4) Let *middle* be a function that maps from any language L over some alphabet Σ to a new language L' as follows:

$middle(L) = \{x: \exists y, z \in \Sigma^* \; (yxz \in L)\}$.

(a) Let $L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* : \#_{\mathtt{a}}(w) = \#_{\mathtt{b}}(w)\}$. What is *middle*($L$)?

$\{\mathtt{a}, \mathtt{b}\}^*$

(b) Prove that, for any language L, if L is context free then $M(L)$ is context free.

The proof is by a construction similar to the one given for *init* except that we build two extra copies of M, both of which mimic all of M's transitions except they read no input. From each state in copy one, there is a transition labeled $\varepsilon/\varepsilon/\varepsilon$ to the corresponding state in M, and from each state in M there is a transition labeled $\varepsilon/\varepsilon/\varepsilon$ to the corresponding state in the second copy. The start state of M^* is the start state of copy 1. So M^* begins in the first copy, performing, without actually reading any input, whatever stack operations M could have performed while reading some initial input string y. At any point, it can guess that it's skipped over all the characters in y. So it jumps to M and reads x. At any point, it can guess that it's read all of x. Then it jumps to the second copy, in which it can do whatever stack operations M would have done on reading z. If it guesses to do that before it actually reads all of x, the path will fail to accept since it will not be possible to read the rest of the input.

(5) Provide short answers to each of the following questions:

(a) Let $L_1 = L_2 \cap L_3$.

(i) Show values for L_1, L_2, and L_3, such that L_1 is context-free but neither L_2 nor L_3 is.

Let: $L_1 = \{\mathtt{a}^n\mathtt{b}^n : n \geq 0\}$.
$L_2 = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^j : j \leq n\}$.
$L_3 = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^j : j = 0 \text{ or } j > n\}$.

(ii) Show values for L_1, L_2, and L_3, such that L_2 is context-free but neither L_1 nor L_3 is.

Let: $L_2 = \mathtt{a}^*$.
$L_1 = \{\mathtt{a}^n : n \text{ is prime}\}$.
$L_3 = \{\mathtt{a}^n : n \text{ is prime}\}$.

(b) Give an example of a regular language L_1 that has a superset L_2 that is context-free but not regular. (Specify both L_1 and L_2 in your example.)

$L_1 = \{\varepsilon\}$. $L_2 = \mathrm{A}^n\mathrm{B}^n$.

(6) Consider the following grammar G:

$$S \rightarrow 1\,S\,1 \mid T$$
$$T \rightarrow 1\,X\,1 \mid X$$
$$X \rightarrow 0\,X\,0 \mid 1$$

(a) What are the first four strings in the lexicographic enumeration of $L(G)$?

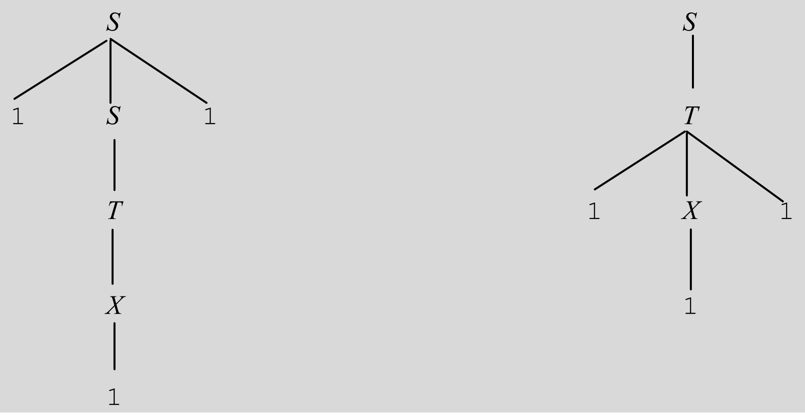
1, 010, 111, 00100

(b) Give an example of a string $w \in \{0, 1\}^+$ such that $|w| > 7$ and $w \notin L(G)$.

00000000, or, for other examples, any string of even length.

(c) Show that G is ambiguous.

The string 111 has more than one parse tree in G:



(7) Give a short English description of what the following machine M does. We are looking for a high-level description of the result of running M, not a play-by-play description of how it works. Assume that the input to M is in $1\,(0 \cup 1)^*$. $M =$

$$\begin{tikzcd}
{>R_{\square}\, L\, L} \arrow[rr, "1"] \arrow[d, "\square"'] \arrow[drr, "0"] & & 0 \arrow[ll, bend right] \\
{1\, L} & & {1\, L_{\square}}
\end{tikzcd}$$

Viewing its input as a binary number, M adds 2.