No books, no calculators. One 8.5x11 page of handwritten notes.

Problem 0: 1 point:
Name: ________________________________

Student ID: ________________________________

Section (circle one): 8AM 1PM 2PM 3PM

<table>
<thead>
<tr>
<th>Problem</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>/1</td>
</tr>
<tr>
<td>1</td>
<td>/30</td>
</tr>
<tr>
<td>2</td>
<td>/24</td>
</tr>
<tr>
<td>3</td>
<td>/15</td>
</tr>
<tr>
<td>4</td>
<td>/30</td>
</tr>
<tr>
<td>5</td>
<td>/10</td>
</tr>
<tr>
<td>Total</td>
<td>/100</td>
</tr>
</tbody>
</table>
1. 30 pts. Prove the following languages (all with input alphabet \{0, 1\}) are regular or not.

(a) Non-empty strings with the last symbol equal the first symbol.
Solution: The language is regular. Here’s a regular expression for the language: (0(0|1)*0)|(1(0|1)*1).

(b) Odd-length strings with the first symbol equal the middle symbol.
Solution: The language is not regular.
Let our adversary choose \( p \) as specified in the Pumping Lemma. We choose \( s = 10^{p-1}1^{p+1} \) which is in the language. Our adversary choose \( x, y, z \) such that \( s = xyz, |y| > 0, |xy| \leq p \). Given that \( y \) must be within the first \( p \) symbols, either \( y \) includes the first one (and \( x \) must be empty), or \( y \) is all zeros. In either case, \( |y| \) must be even, since otherwise we’d pump to an even-length string. In the first case, \( xy^0z \) begins with a 0 and has a middle symbol of 1, so is not in the language. In the second case, \( xy^3z \) gives us a string of the form \( 10^q1^{p+1} \) with \( q > p \). In this case, the first symbol is 1 and the middle symbol is 0.
Thus, we’ve shown that the pumping lemma doesn’t hold for this language. But since the pumping lemma holds for all regular languages, this language is not regular.
2. 24 pts. Consider the following NFA:

(a) Convert the NFA to a minimal DFA.

Solution:

This DFA is already minimal. \( \{A, C\} \) is distinguished from the other two since it is accepting. On input symbol 1, \( \{A\} \) goes to an accepting state, while \( \{A, B\} \) goes to a non-accepting state. Thus, they are distinguished.

(b) Give a regular expression that generates the same language the automata accepts.

Solution:

You can start with the original NFA, or with the DFA. Here we’ll start with the DFA: First, convert to a GNFA:
Next, remove the \( \{A, B\} \) state:

Then, remove the \( \{A, C\} \) state:

Finally, remove the \( \{A\} \) state:
This yields a GNFA with one start state, one accepting state, and a transition of $0^*(1|(0(0|1)))^*$.
3. 15 pts. Convert the following grammar to Chomsky Normal Form (show your work):

\[
S \rightarrow ASB | \varepsilon \\
A \rightarrow aAS | a \\
B \rightarrow SbS | A | bb
\]

**Solution:** First, let’s remove \(\varepsilon\)-productions. We’ll add a new start state to account for the fact that \(S\) can generate \(\varepsilon\). We’ll start with removing \(S \rightarrow \varepsilon\).

\[
S' \rightarrow S | \varepsilon \\
S \rightarrow ASB | AB \\
A \rightarrow aAS | aA | a \\
B \rightarrow SbS | Sb | bb
\]

No further \(\varepsilon\)-productions exists, so we’re done with this step.

Next, let’s remove unit productions \((B \rightarrow A)\):

\[
S' \rightarrow ASB | B | \varepsilon \\
S \rightarrow ASB | AB \\
A \rightarrow aAS | aA | a \\
B \rightarrow SbS | Sb | bb
\]

Now, let’s add new non-terminals for the terminals that occur not by themselves:

\[
S' \rightarrow ASB | B | \varepsilon \\
S \rightarrow ASB | AB \\
A \rightarrow YAS | YA | a \\
B \rightarrow SZS | ZS | b | YAS | YA | a | ZZ \\
Y \rightarrow a \\
Z \rightarrow b
\]
Finally, let’s modify productions with more than two non-terminals on the RHS:

\[
\begin{align*}
S' & \rightarrow AC|AB|\varepsilon \\
S & \rightarrow AC|AB \\
A & \rightarrow YD|YA|a \\
B & \rightarrow SE|ZS|SZ|b|YD|YA|a|ZZ \\
C & \rightarrow SB \\
D & \rightarrow AS \\
E & \rightarrow ZS \\
Y & \rightarrow a \\
Z & \rightarrow b 
\end{align*}
\]

This grammar is in Chomsky Normal Form.
4. 30 pts.  For each language, decide whether it is context-free, non-context-free, or deterministic context-free. Prove your answer. (Hint: there’s one of each.)

(a) \{0^i1^j2^k| i, j \geq 0\}

Solution: This language is not context-free. We’ll use the pumping lemma to prove it.

Let our adversary choose \( p \). We’ll choose \( s = 0^p1^p2^p3^p \). Our adversary choose \( v,w,x,y,z \) such that \( s = vwxyz, |wxy| \leq p, |wy| > 0 \). Given the restriction on its length, \( wxy \) contains at most two different symbols from the input alphabet: 0 and 1, 1 and 2, or 2 and 3. Consider \( s' = vw^2xy^2z \). Since either \( w \) or \( y \) is non-empty, \( s' \) has one or two symbols from the input alphabet that occur more than \( p \) times. However, if 0 is one of those symbols, 2 is not, so the string has more 0’s than 2’s. Similar arguments hold for the other 3 symbols. Thus, \( s' \) is not in the language.

Therefore, the pumping lemma does not hold for this language. But, since the pumping lemma holds for all context-free languages, this language is not context-free.

(b) \{0^i1^j2^k| i, j \geq 0 and (i = 3j or j = 3k)\}

Solution: This language is context-free. Here’s a context-free grammar for it:

\[
S \rightarrow AT | ZB  \\
A \rightarrow 000A1 | \varepsilon \quad \text{generates } 0^i1^j, i = 3j  \\
B \rightarrow 111B2 | \varepsilon \quad \text{generates } 1^j2^k, j = 3k  \\
T \rightarrow 2T | \varepsilon \quad \text{generates } 2^*  \\
Z \rightarrow 0Z | \varepsilon \quad \text{generates } 0^*  
\]

(c) The complement of \{0^i1^j| i, j \geq 0 and i = 3j\}

Solution: This language is deterministic context-free. Here’s a narrative description of a deterministic PDA for \( L^c = \{0^i1^j| i, j \geq 0 \text{ and } i = 3j\} \):

- Push a $ and go to a pushing state.
- While in the pushing state, for each 0, push 3A symbols.
• When a 1 is read, pop an A symbol from the stack and switch to a popping state.
• While in the popping state, for each 1, pop an A symbol.
• When a $ is read, go to an accepting state.

Since $L^c$ is deterministic context-free and since deterministic context-free languages are closed under complement, the original language is deterministic context-free.
5. 10 pts. Extra Credit. Circle all of the following that are correct:

(a) Since the pumping lemma applies to all context-free languages, all context-free languages contain an infinite number of strings.
\textit{Solution:} False. The empty language is a context-free language which has a finite number of strings. The pumping lemma says that strings greater than some particular (language-specific) length are pumpable. If no strings are greater than that length, the pumping lemma is vacuously true.

(b) A pushdown automata is basically a finite state automata with an extra stack.
\textit{Solution:} True

(c) The stack alphabet of a pushdown automata must be the same as the input alphabet.
\textit{Solution:} False. It can be different.

(d) The stack alphabet of a pushdown automata must be different from the input alphabet.
\textit{Solution:} False. It can be the same.

(e) Finite-state automata can’t count.
\textit{Solution:} Ambiguous question. True, they can’t count more than some fixed amount. False, they can count a fixed amount. Either answer scored.

(f) Pushdown automata can count one thing at a time.
\textit{Solution:} True. For example, $a^nb^mc^nd^m$ can’t be recognized by a PDA because it can’t count both the $n$ and the $m$ simultaneously.

(g) A context-free grammar can have an infinite set of rules (productions).
\textit{Solution:} False

(h) A context-free grammar in Chomsky Normal Form derives a string of length $n$ in exactly $2^n - 1$ steps.
\textit{Solution:} False. It’s $2n - 1$ steps.

(i) If a grammar has two different parse trees for the same string, the grammar is ambiguous.
\textit{Solution:} True.
(j) If a grammar has two different derivations for the same string, the grammar is ambiguous.

*Solution:* False. If a grammar has two *leftmost* derivations for the same string (each of which generates a distinct parse tree), then the grammar is ambiguous. Even non-ambiguous grammars often have two derivations for the same string.