1 Regular Expressions and Finite Automata

Consider a small language using only the letters “z”, “o”, and the slash character “/”. A comment in this language starts with “/o” and ends after the very next “o/”.

(a) Give a regular expression that matches exactly one complete comment and nothing else. Assume comments do not nest. For full credit, use only the core notations: ε, “ab”, AB, A|B, and A\* [8 points]

Consider the language over the alphabet $\Sigma = \{a, b\}$ containing strings in which number of 'a's is a multiple of 3
or number of 'b's is a multiple of 2.

(b) Write a regular expression for this language. [4 points]

(c) Write an NFA for this language. [4 points]

(d) Write a DFA with at most 6 states for this language [4 points]

(3) Can we construct a regular expression for a language over the alphabet \( \Sigma = \{a, b\} \) whose strings have equal number of occurrences of a and b? Explain. [4 points]
2 LL Parsing

Consider the following grammar with terminals *, !, n, (, and ).

\[
\begin{align*}
E & \rightarrow \ F \ H \\
H & \rightarrow \ ^* \ E \ | \ \epsilon \\
F & \rightarrow \ F \ ! \ | \ G \\
G & \rightarrow \ n \ | \ (E)
\end{align*}
\]

(a) The grammar is not LL(1). Explain in one sentence why [2 points]

(b) Fix the grammar to make it LL(1) by filling in the blanks below [4 points]

\[
\begin{align*}
E & \rightarrow \ F \ H \\
F & \rightarrow \ ________ \\
H & \rightarrow \ ^* \ E \ | \ \epsilon \\
G & \rightarrow \ n \ | \ (E)
\end{align*}
\]

(c) Compute the first and the follow set of the fixed grammar [8 points]

<table>
<thead>
<tr>
<th></th>
<th>First</th>
<th>Follow</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(d) Compute LL(1) parsing table [10 points]

<table>
<thead>
<tr>
<th></th>
<th>*</th>
<th>!</th>
<th>(</th>
<th>)</th>
<th>$</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3 LR Parsing and Ambiguity

Consider the following grammar with terminals - (the negation operator) and \textit{int}.

\[
\begin{align*}
S & \rightarrow \ E \\
E & \rightarrow \ E - E \ | \ - - E \ | \ \textit{int}
\end{align*}
\]

(a) Draw all the parse trees for the string \textit{int} - - \textit{int} - \textit{int} [6 points]
(b) Is this grammar ambiguous? Why or why not? [2 point]
(c) Complete the above partial LR(1) DFA for the grammar. [16 points]
- Fill in items of all states by performing closure operation. (6)
- Fill in missing transition labels on all edges (4)
- Write the necessary “reduce by … on … ” labels on states (2)
- Add missing transition edges (Hint: State 2 and State 5) (4)

(d) For each state with a conflict, list the state, the lookahead token, and the type of conflict (i.e. shift-reduce conflict, or reduce-reduce conflict). [4 points]

Suppose we want the string - - int - int - int to have only the following parse tree (call this property P).
(c) Describe in English the precedence and associativity rules necessary to ensure property P. [4 points]

(f) Explain, for each conflict in the LR(1) parsing DFA for this grammar, how it should be resolved to ensure property P. [4 points]

(g) Rewrite the grammar to an equivalent unambiguous grammar to ensure property P. Two grammars are equivalent when they accept the same language. [8 points]

(h) The Cool grammar has an ambiguity introduced by let-expression. Give an example illustrating the ambiguity associated with let-expression. [4 points]