Please read all instructions (including these) carefully.

This is a closed-book exam. You are allowed a one-page handwritten cheat sheet.

Write your name, login, and SID.

There are TODO pages in this exam and 3 questions, each with multiple parts. If you get stuck on a question move on and come back to it later.

You have 1 hour and 15 minutes to work on the exam.

Please write your answers in the space provided on the exam, and clearly mark your solutions. You may use the backs of the exam pages as scratch paper. Do not use any additional scratch paper.

Solutions will be graded on correctness and clarity. Each problem has a relatively simple and straightforward solution. Partial solutions will be graded for partial credit.

No electronic devices are allowed, including cell phones used merely as watches. Silence your cell phones and place them in your bag.

LOGIN: ______________________________________________

NAME: ______________________________________________

SID: ______________________________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>Max points</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td><strong>Sub Total</strong></td>
<td><strong>100</strong></td>
<td></td>
</tr>
</tbody>
</table>

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]

/o((o*z))/|o*o/

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]

\((a^*ba^*ba^*)^* \mid (b^*ab^*ab^*a^*b^*)^*\)

(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]

No. Corresponding state machine requires infinite number of states.

Yes/No (1)
Reason (3)
2 LL Parsing

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

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

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

\[F \rightarrow F !\] is left recursive

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

\[
\begin{align*}
E & \rightarrow F H \\
H & \rightarrow * E \mid \epsilon \\
F & \rightarrow G K \\
G & \rightarrow n \mid (E)
\end{align*}
\]

(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>(, n</td>
<td>$, )</td>
</tr>
<tr>
<td>F</td>
<td>(, n</td>
<td>$, ), *</td>
</tr>
<tr>
<td>G</td>
<td>(, n</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>$</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>E \rightarrow FH</td>
<td>E \rightarrow FH</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F \rightarrow GK</td>
<td>F \rightarrow GK</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>G \rightarrow (E)</td>
<td>G \rightarrow n</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>H \rightarrow *E</td>
<td>$</td>
<td>$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>$</td>
<td>K \rightarrow !K</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}.

\[
S \rightarrow E \\
E \rightarrow E - E \mid - - E \mid \textit{int}
\]

(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]

The grammar is ambiguous because it admits multiple parse trees for string `int -- int -- int`
(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]

3: Lookahead token: -, shift-reduce conflict
4: Lookahead token: -, shift-reduce conflict

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

1. "-" is right-associative
2. "--" has higher precedence than "-"

(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]

State 3: reduce instead of shift on "-"
State 4: shift instead of reduce on "-"

(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]

S → E  
E → T - E | T  
T → -- T | int

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

let ... in 1 + 2
can be parsed as:
(let … in 1) + 2

or

let … in (1 + 2)