EXAM

Please read all instructions, including these, carefully

- There are 7 questions on the exam, with multiple parts. You have 3 hours to work on the exam.
- The exam is open book, open notes.
- Please write your final answers in the space provided on the exam. You may use the backs of the exam pages as scratch paper, or use additional pages (available at the front of the room).
- Each problem has a straightforward solution. Solutions will be graded on correctness and clarity. Partial solutions will be given partial credit.

NAME: ______________________________________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>Max points</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
1. Consider the following augmented CFG for prefix expressions.

\[
E' \rightarrow E \\
E \rightarrow + E E \mid - E E \mid \text{id}
\]

(a) Compute the following closures of items.

i) \(I_a = \text{closure}(\{E' \rightarrow \bullet E\})\)

\[
\{ E \rightarrow \bullet E \\
E \rightarrow \bullet + E E \\
E \rightarrow \bullet - E E \\
E \rightarrow \bullet \text{id} \}
\]

ii) \(I_b = \text{closure}(\{E \rightarrow + \bullet EE\})\)

\[
\{ E \rightarrow + \bullet EE \\
E \rightarrow \bullet + E E \\
E \rightarrow \bullet - E E \\
E \rightarrow \bullet \text{id} \}
\]

iii) \(I_c = \text{closure}(\{E \rightarrow +E \bullet E\})\)

\[
\{ E \rightarrow + E \bullet E \\
E \rightarrow \bullet + E E \\
E \rightarrow \bullet - E E \\
E \rightarrow \bullet \text{id} \}
\]

iv) \(I_d = \text{closure}(\{E \rightarrow \text{id} \bullet\})\)

\[
\{ E \rightarrow \text{id} \bullet \}
\]

(b) Show the following entries of the SLR(1) parsing table \(M\), where \(I_b\) and \(I_d\) indicate the states corresponding to the items sets computed in (a).

i. \(M[I_b, E] \quad [\text{goto}] \ I_c\)

ii. \(M[I_d, +] \quad \text{Reduce } E \rightarrow \text{id}\)

iii. \(M[I_d, \$] \quad \text{Reduce } E \rightarrow \text{id}\)
2. Consider extending the LR(1) case to the general $k$ lookahead symbols. Given an LR($k$) grammar with $N$ nonterminals and $T$ terminals, how many columns would we have in a corresponding LR($k$) action/goto table? State your assumptions and show your work for full credit.

With one symbol lookahead, we have one column for each terminal and one for each non-terminal, so $N + T$ columns. For $k$ lookahead, we still have one column for each non-terminal and each individual terminal, but a column must be added for each possible combination of $k$ symbols. So when $k = 2$, we have $T + T(T-1) + N$ columns. For larger values of $k$,
3. Consider the following grammar \( G \).

\[
\begin{align*}
E & \rightarrow E + T \mid T \\
T & \rightarrow \text{id} \mid \text{id}( ) \mid \text{id}( L ) \\
L & \rightarrow E ; L \mid E
\end{align*}
\]

(a) Give an LL(1) grammar that generates the same language as \( G \).

First, eliminate left recursion:

\[
\begin{align*}
E & \rightarrow TE' \\
E' & \rightarrow + TE' \mid \varepsilon \\
T & \rightarrow \text{id} \mid \text{id}( ) \mid \text{id}( L ) \\
L & \rightarrow E ; L \mid E
\end{align*}
\]

Then eliminate common prefixes by left factoring:

\[
\begin{align*}
E & \rightarrow TE' \\
E' & \rightarrow + TE' \mid \varepsilon \\
T & \rightarrow \text{id} T' \\
T' & \rightarrow ( T'' \mid \varepsilon \\
T'' & \rightarrow ) \mid L ) \\
L & \rightarrow E L' \\
L' & \rightarrow ; L \mid \varepsilon
\end{align*}
\]

(b) Prove that your grammar is LL(1) by whatever method you like. (Hint: This does not require an exhaustive construction.) Don’t just restate the definition of LL(1).

Remember a grammar \( G \) is LL(1) iff whenever \( A \rightarrow u \mid v \) are two distinct productions of \( G \), the following conditions hold:

- For no terminal \( a \) do both \( u \) and \( v \) derive strings beginning with \( a \).
- At most one of \( u \) and \( v \) can derive the empty string (\( \varepsilon \)), and
- If \( v \Rightarrow^* \varepsilon \) then \( u \) does not derive any string beginning with an element in \( \text{FOLLOW}(A) \).

By constructing these FIRST and FOLLOW sets, we can prove the three requirements are satisfied:

\[
\begin{align*}
\text{FIRST}( + TE') = \{ + \} \\
\text{FOLLOW}( E') = \{ \$, \), ; \} \\
\text{FIRST}( T' ) = \{ ( \} \\
\text{FOLLOW}( T' ) = \{ +, \$, \} \\
\text{FIRST}( L ) = \{ \text{id} \} \\
\text{FOLLOW}( L' ) = \{ \} \\
\text{FIRST}( ; L ) = \{ ; \}
\end{align*}
\]
$E'$, $T'$, $T''$, and $L'$ are the only nonterminals with two productions.

1. Only $T''$ has 2 rules deriving non-null strings, and their FIRST sets are distinct.
2. At most of one rule of $E'$, $T'$ and $L'$ derives $\epsilon$, since the other always includes a terminal.
3. $E'$, $T'$ and $L'$ are nullable, but the non-null rule of each RHS doesn’t derive any string beginning with a symbol in the FOLLOW set of the LHS.

4. Here is an attribute grammar $G$ for arithmetic expressions using multiplication, unary -, and unary +.

\[
S \rightarrow E \{ \text{if } (E.sign == \text{POS}) \text{ print(“result is positive”);} \\
\text{else print(“result is negative”);} \}
\]

\[
E \rightarrow \text{unsigned_int} \\
| + E \\
| - E \\
| E \cdot E 
\]

(a) Add semantic actions to compute an attribute $E.sign$ for non-terminal $E$ to record whether the arithmetic value of $E$ is positive or negative. The attribute sign can have two values, either POS or NEG. To get you started, the first rule is provided.

\[
E \rightarrow \text{unsigned_int} \{ E.sign = \text{POS} ; \} \\
| + E_1 \{ E.sign = \text{E}_1.sign ; \} \\
| - E_1 \{ \text{if } (E_1.sign == \text{POS}) \\
\text{E.sign = NEG ;} \\
\text{else } E.sign = \text{POS} ; \}
| E_1 \cdot E_2 \{ \text{if } (E_1.sign == E_2.sign) \\
E.sign = \text{POS} ; \\
\text{else } E.sign = \text{NEG} ; \}
\]

(b) Show the parse tree for input $2 \cdot - 3$ (where 2 and 3 are “unsigned_ints”). Indicate at each node what the values of associated attributes are.
(c) Is $E$.sign an inherited attribute or a synthesized attribute?

Synthesized, as it is propagated from the leaf nodes upwards through the tree.
5. Consider the following grammar $G$:

$S \rightarrow Sa \mid bL$
$L \rightarrow ScL \mid Sd \mid a$

(a) Remove left recursion from $G$. The result is called $G'$.

$S \rightarrow bLS'$
$S' \rightarrow aS' \mid \epsilon$
$L \rightarrow ScL \mid Sd \mid a$

(b) Left factor $G'$. The result is $G''$.

$S \rightarrow bLS'$
$S' \rightarrow aS' \mid \epsilon$
$L \rightarrow SL' \mid a$
$L' \rightarrow cL \mid d$

(c) Fill in the table below with the FIRST and FOLLOW sets for the non-terminals in grammar $G''$ (note that the number of rows may be more than you need):

<table>
<thead>
<tr>
<th>Non-terminal</th>
<th>First</th>
<th>FOLLOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>$b$</td>
<td>$c, d, $</td>
</tr>
<tr>
<td>$S'$</td>
<td>$a, \epsilon$</td>
<td>$c, d, $</td>
</tr>
<tr>
<td>$L$</td>
<td>$a, b$</td>
<td>$a$</td>
</tr>
<tr>
<td>$L'$</td>
<td>$c, d$</td>
<td>$a$</td>
</tr>
</tbody>
</table>

(d)

(e)

(f) Fill in the LL(1) parse table for this grammar.
### CS331 Compiler Design
Spring 2006

<table>
<thead>
<tr>
<th>Non-terminal</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
<th>$$$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td></td>
<td>$S \rightarrow bLS'$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S'$</td>
<td></td>
<td>$S' \rightarrow aS'$</td>
<td>$S' \rightarrow \varepsilon$</td>
<td>$S' \rightarrow \varepsilon$</td>
<td>$S' \rightarrow \varepsilon$</td>
</tr>
<tr>
<td>$L$</td>
<td>$L \rightarrow a$</td>
<td>$L \rightarrow SL'$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$L'$</td>
<td></td>
<td>$L' \rightarrow cL$</td>
<td>$L' \rightarrow d$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(e) Using the parse table in (d), show a top-down parse of the string $bbacbada$$.$

<table>
<thead>
<tr>
<th>Stack (with top at left)</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>S$</td>
<td>bbacbada$$</td>
<td>$S \rightarrow bLS'$</td>
</tr>
<tr>
<td>bLS'$</td>
<td>bbacbada$$</td>
<td>match $b$</td>
</tr>
<tr>
<td>LS'$</td>
<td>bacbada$$</td>
<td>$L \rightarrow SL'$</td>
</tr>
<tr>
<td>SL'$</td>
<td>bacbada$$</td>
<td>$S \rightarrow bLS'$</td>
</tr>
<tr>
<td>bLS'L'S'$</td>
<td>bacbada$$</td>
<td>match $b$</td>
</tr>
<tr>
<td>LS'L'S'$</td>
<td>acbada$$</td>
<td>$L \rightarrow a$</td>
</tr>
<tr>
<td>aS'L'S'$</td>
<td>acbada$$</td>
<td>match $a$</td>
</tr>
<tr>
<td>S'L'S'$</td>
<td>cbada$$</td>
<td>$S' \rightarrow \varepsilon$</td>
</tr>
<tr>
<td>L'S'$</td>
<td>cbada$$</td>
<td>$L' \rightarrow cL$</td>
</tr>
<tr>
<td>cLS'$</td>
<td>cbada$$</td>
<td>match $c$</td>
</tr>
<tr>
<td>LS'$</td>
<td>bada$$</td>
<td>$L \rightarrow SL'$</td>
</tr>
<tr>
<td>SL'$</td>
<td>bada$$</td>
<td>$S \rightarrow bLS'$</td>
</tr>
<tr>
<td>bLS'L'S'$</td>
<td>bada$$</td>
<td>match $b$</td>
</tr>
<tr>
<td>LS'L'S'$</td>
<td>ada$$</td>
<td>$L \rightarrow a$</td>
</tr>
<tr>
<td>aS'L'S'$</td>
<td>ada$$</td>
<td>match $a$</td>
</tr>
<tr>
<td>S'L'S'$</td>
<td>da$$</td>
<td>$S' \rightarrow \varepsilon$</td>
</tr>
<tr>
<td>L'S'$</td>
<td>da$$</td>
<td>$L' \rightarrow d$</td>
</tr>
<tr>
<td>dS'$</td>
<td>da$$</td>
<td>match $d$</td>
</tr>
<tr>
<td>S'$</td>
<td>a$</td>
<td>$S' \rightarrow aS'$</td>
</tr>
<tr>
<td>aS'$</td>
<td>a$$</td>
<td>match $a$</td>
</tr>
<tr>
<td>S'$</td>
<td>$$$</td>
<td>$S' \rightarrow \varepsilon$</td>
</tr>
<tr>
<td>$$$</td>
<td>$$$</td>
<td>ACCEPT</td>
</tr>
</tbody>
</table>
6. Consider the following simple context free grammars:

\[
G_1: \quad \begin{align*}
S & \to Aa \\
A & \to \varepsilon \\
A & \to bAb
\end{align*}
\]

\[
G_2: \quad \begin{align*}
S & \to Aa \\
A & \to \varepsilon \\
A & \to Abb
\end{align*}
\]

Note that the grammars generate the same language: strings consisting of even numbers of b’s (including 0 of them), followed by an a.

(a) Attempt to show a shift-reduce parse of the string bbbba for a parser for grammar \(G_1\). Show the contents of the stack, the input, and the actions (i.e., shift, reduce, error, accept). You don’t need to create a parse table; just use your knowledge of the grammar and how the parser works. Be sure to indicate any conflicts and explain why they are conflicts. Is \(G_1\) LR(1)? Is \(G_1\) LR(0)?

<table>
<thead>
<tr>
<th>Stack (with top at right)</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>bbbba$</td>
<td>shift b</td>
</tr>
<tr>
<td>$b$</td>
<td>bbbab$</td>
<td>shift b</td>
</tr>
<tr>
<td>$bb$</td>
<td>bba$</td>
<td>CONFLICT: reduce A \to \varepsilon or shift b?</td>
</tr>
</tbody>
</table>

There is always a conflict because

(b) Attempt to show a shift-reduce parse of the string bbbba for a parser for grammar \(G_2\). Show the contents of the stack, the input, and the actions (i.e., shift, reduce, error, accept). You don’t need to create a parse table; just use your knowledge of the grammar and how
the parser works. Be sure to indicate any conflicts and explain why they are conflicts. Is $G_2$ LR(1)? Is $G_2$ LR(0)?
(c) Indicate whether $G_1$ and $G_2$ are LL(1) and explain your answer. You don’t need to construct the parse tables, but may argue from other properties.

(d) Of the language classes we have discussed, which is the smallest category into which $L(G_1)$ fits? Justify your answer.
7. Consider the following GOTO graph of an LALR(1) parser. Notice the state numbers and refer to them in the answers to the following questions.

(a) Which states of the parser have conflicts? What kind of conflicts are they, and under what circumstances do they arise?
(b) Describe the possible contents of the stack when the parser is in state 5.

(c) Give a string that could remain in the input when the parser is in state 5 that would lead to a successful parse.