ECE 573 — Midterm 1
February 22, 2011

Name: ______________________________________

Purdue email: __________________________________

Please sign the following:
I affirm that the answers given on this test are mine and mine alone. I did not receive help from any person or material (other than those explicitly allowed).

X __________________________________________

<table>
<thead>
<tr>
<th>Part</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>100</td>
</tr>
</tbody>
</table>
Part 1: Short answers (10 points)

1) You’ve started working for a new microprocessor company, AMTel, that is working on a new processor based on a VLIW (very large instruction word) architecture. In a VLIW machine, it is the compiler’s responsibility to schedule instructions to take advantage of instruction-level parallelism. Your job is to design the compiler for this architecture. The company only has one kind of architecture. Would you use a standard compiler, or a just-in-time compiler? Justify your answer. (4 points).

I accepted any answer with a good justification. The answer I was looking for was a standard compiler, as portability is not an issue with one architecture, and a standard compiler can spend longer scheduling its instructions.

2) AMTel was wildly successful with their original architecture, and now wants to develop another VLIW processor, but this time with a different set of constraints for when instructions can be scheduled. The company wants their customers to be able to write programs that run, efficiently, on either their new architecture or their old one. Would you use a standard compiler, or a just-in-time compiler? Justify your answer. (4 points).

Here, the portability requirements argue for a just-in-time compiler.

3) Now AMTel wants you to adapt your compiler (from question 2) to work with a new programming language. Which phases of the compiler will you have to rewrite? (2 points).

The front-end needs to be re-written: scanner, parser and semantic routines.
Part 2: Regular expressions, finite automata and scanners (15 points)

1) Describe, in one sentence, the strings captured by the following regular expression. Give a different regular expression that captures the same set of strings (2 points):

\[(xz)\|yz\] \(^*\)

A string consisting of 0 or more repetitions of xz and yz. An equivalent regular expression is: \((xlyz)^*\)

2) Give a non-deterministic FA that accepts strings defined by the following regular expression (5 points):

\[(ab^+c)\|a^+bc\]

3) Use the subset construction on the NFA you created above to generate the transition table for the deterministic version of the automaton (5 points)

<table>
<thead>
<tr>
<th>state</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>final?</th>
<th>new name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 1, 5</td>
<td>2, 6</td>
<td>err</td>
<td>err</td>
<td>n</td>
<td>A</td>
</tr>
<tr>
<td>2, 6</td>
<td>6</td>
<td>3, 7</td>
<td>err</td>
<td>n</td>
<td>B</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>7</td>
<td>err</td>
<td>n</td>
<td>C</td>
</tr>
<tr>
<td>3, 7</td>
<td>err</td>
<td>3</td>
<td>4, 8, 9</td>
<td>n</td>
<td>D</td>
</tr>
<tr>
<td>4, 8, 9</td>
<td>err</td>
<td>err</td>
<td>err</td>
<td>y</td>
<td>E</td>
</tr>
<tr>
<td>7</td>
<td>err</td>
<td>err</td>
<td>8, 9</td>
<td>n</td>
<td>F</td>
</tr>
<tr>
<td>3</td>
<td>err</td>
<td>3</td>
<td>4, 9</td>
<td>n</td>
<td>G</td>
</tr>
<tr>
<td>8, 9</td>
<td>err</td>
<td>err</td>
<td>err</td>
<td>y</td>
<td>H</td>
</tr>
<tr>
<td>4, 9</td>
<td>err</td>
<td>err</td>
<td>err</td>
<td>y</td>
<td>I</td>
</tr>
</tbody>
</table>

4) Give the reduced version of the DFA you created in step 3, in graphical form. (3 points):
Part 3: Grammars (10 points)

Let G be the grammar:

\[
\begin{align*}
S & \rightarrow AB \\
A & \rightarrow Ax \\
A & \rightarrow \lambda \\
B & \rightarrow yB \\
B & \rightarrow y
\end{align*}
\]

Using this grammar, answer the following questions.

1) What are the terminals and non-terminals of this grammar? (1 point)

 terminals: x, y (not lambda!)  
 non-terminals: S, A, B

2) Draw the parse tree for the string “xxy” (4 points)

3) Can the language of this grammar be captured by a regular expression? If so, give the regular expression. If not, give a short argument why not. (6 points)

Yes: x*y+
Part 4: LL parsers (20 points)

Answer the questions in this part using the following grammar:

1. S → AxB
2. A → y
3. A → Bz
4. B → λ
5. B → zA

1) Give the First sets for each non-terminal in the grammar (6 pts)

First(S) = {z, y}
First(A) = {z, y}
First(B) = {z, λ}

2) Give the Follow sets for each non-terminal in the grammar (6 pts)

Follow(S) = { }
Follow(A) = {x, z}
Follow(B) = {z}

3) Give the parse table for this grammar (6 pts)

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>err</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>err</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>err</td>
<td>err</td>
<td>4, 5</td>
</tr>
</tbody>
</table>

4) Is this grammar LL(1) or not? Why or why not? (2 pts)

No, there is a conflict in the parse table. When we are trying to parse “B” and see a ‘z’, we do not know which rule to predict.
Part 5: LR(0) Parsers (25 points)

Use the following grammar for the next two questions:

1. $S \rightarrow AB$
2. $A \rightarrow (Ab)$
3. $A \rightarrow (A)$
4. $B \rightarrow xB$

1) Fill in the missing states for the following CFSM (19 points) and fill in the missing edge labels (1 point)

2) List the reduce states in the above CFSM, and the shift states (3 points)

Reduce: 3, 6, 9, 10
Shift: the rest

3) Is this an LR(0) grammar? Why or why not? (2 points)

Yes, there are no shift/reduce or reduce/reduce conflicts.
Part 6: LR(1) Parsers (20 points)

Consider the grammar:

1. $S \rightarrow AB$
2. $A \rightarrow xB$
3. $A \rightarrow \lambda$
4. $B \rightarrow zA$

1) Fill in the missing states in the LR(1) machine given below (15 points)

2) Show the sequence of actions taken when the LR(1) parser parses the string “xzxz.” Use actions of the format “Shift X,” “Reduce R (goto X),” and “Accept.” Assume the parser accepts when it gets to State 2. (5 points)

S3, S8, S3, S8, R3(goto 7), R4(goto 6), R2(goto 7), R4(goto 6), R2(goto 1), S4, R3(goto 5), R4(goto 2), Accept