ECE 468/573 — Midterm 1

September 30, 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 _____

Part	Points	Score
1	10	
2	15	
3	15	
4	25	
5	35	
6	20	
Total	120	

Part 1: Short answers (10 points)

1) You are writing a compiler that can produce two different kinds of code: x86 assembly code or Java bytecode (which will run on any Java Virtual Machine), and the user can select which type of code she wants at compile time. Give a scenario where she would choose to generate assembly code (i.e., give a scenario where generating assembly code would be better than producing byecode). Assume that a JVM is available on her target machine. (4 points).

I would rather generate assembly when it takes a long time to compile for a particular target architecture, or when the program is complex and may take a long time to compile

2) Give a scenario where the programmer would choose to produce bytecode instead of assembly. Assume the target machine has an x86 architecture. (4 points).

If the final program is meant to be run on many architectures, or if the behavior of the program is very different for different inputs (so I may want to optimize it differently), I would prefer to generate bytecode.

3) To produce this kind of compiler, name the compiler passes that are *common* for both the x86 and the bytecode versions. (2 points).

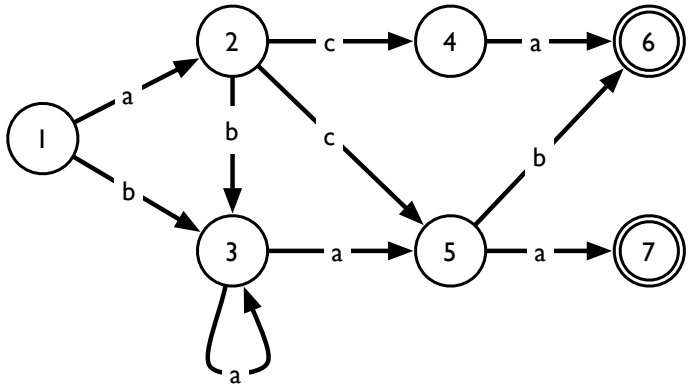
Common passes: scanner, lexer, semantic actions. Code generation and some of the optimization passes will have to be different.

Part 2: Regular expressions, finite automata and scanners (15 points)

A *well-parenthesized string* follows three rules: (i) The only characters are "(" and ")". (ii) There must be an equal number of "(" and ")" in the string. (iii) In any *prefix* of the string, there cannot be more ")" than "(".
1) I want to build a finite automaton to capture well-parenthesized strings. Is this possible? Why or why not? (5 points):

Not possible: to satisfy rule ii, we would have to count how many "(" there are to make sure that we get an equal number of ")"s

2) Consider the following NFA. Fill in the transition table below with its corresponding DFA using the subset construction. (8 points):



State	Final?	a	b	c	New Name
1	N	2	3	—	1
2	N	—	3	4, 5	2
3	N	3, 5	—	—	3
4, 5	N	6, 7	6	—	4
3, 5	N	3, 5, 7	6	—	5
6, 7	Y	—	—	—	6
6	Y	—	—	—	7
3, 5, 7	Y	3, 5, 7	6	—	8

1 point per line. Partial credit given.

3) List which states should be merged when you reduce the above DFA (2 points):
Merge new states 6 and 7. Note that new states 8 and 5 have the same transitions, but 8 is final and 5 is not, so we cannot merge them. -1 point for listing 8&5

Part 3: Grammars (15 points)

Let G be the grammar:

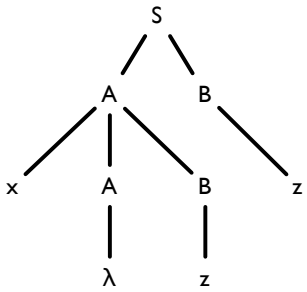
$$
\begin{aligned}
S &\rightarrow AB\$ \\
A &\rightarrow xAB \\
A &\rightarrow yAB \\
A &\rightarrow \lambda \\
B &\rightarrow z
\end{aligned}
$$

Using this grammar, answer the following questions.

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

Terminals: x, y, z also accepted $
Non-terminals: S, A, B

2) Draw the parse tree for the string "xzz" (4 point)



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. (5 points)

No: this grammar requires that if we have *i* xs, and *j* ys, we must have (*i + j + 1*) zs, and we can't do that with an FA. Thus, we can't do this with a regular expression.

4) Give a grammar that captures the matched-parentheses strings from problem 2.1. (5 points)

See the grammar in part 5, but with the "A → x" rule replaced with "A → λ"
-3 points if (^i)^i grammar given
-1 point if grammar "close" (i.e., more than nested parentheses, only a little off from correct)

Part 4: LL parsers (25 points)

Answer the questions in this part using the same grammar from Part 3.

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

First(S) = x, y, z
First(A) = x, y, λ
First(B) = z

2 points per set

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

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

2 points per set

3) Fill in the following parse table (12 pts)

	x	y	z	$
S	1	1	1	
A	2	3	4	
B			5	

1 point per box

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

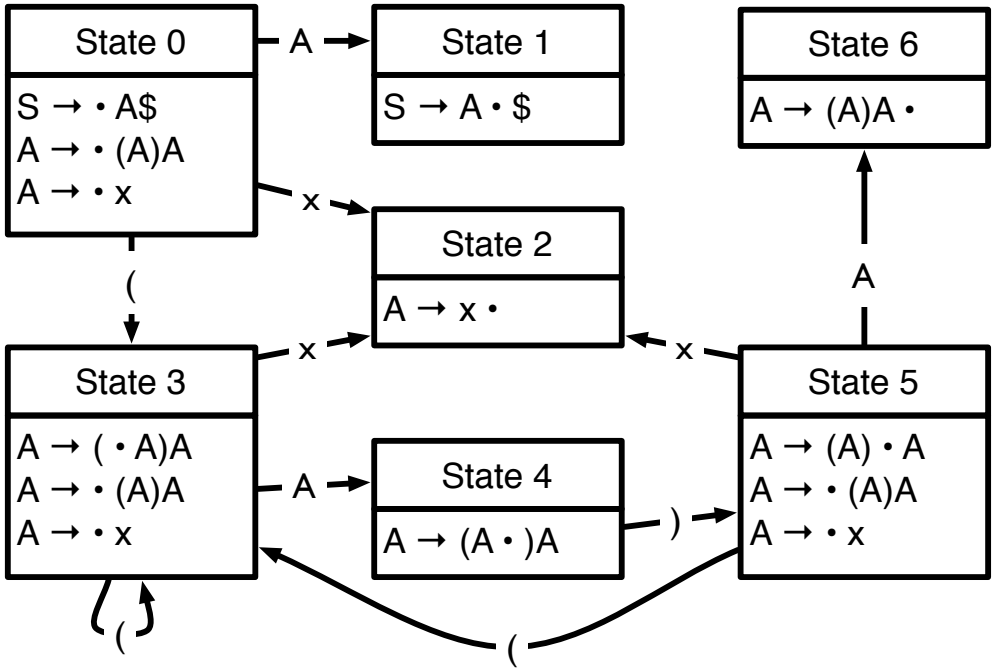
Yes, this is LL(1)—there is no conflict in the parse table.

Part 5: LR(0) Parsers (35 points)

Use the following grammar for the next two questions:

$$\begin{array}{lcl} S & \rightarrow & A\$ \\ A & \rightarrow & (A)A \\ A & \rightarrow & x \end{array}$$

1) Fill in the missing information for the for the following CFSM (10 points)



-1 point per missing configuration

2) List the reduce states in the above CFSM, and the shift states (5 points)

Reduce: 2, 6
Shift: 0, 3, 4, 5
-1 point if state 1 listed as reduce state. OK if listed as shift state

3) Is this an LR(0) grammar? Why or why not? (4 points)

Yes, this is an LR(0) grammar: none of the states have shift/reduce or reduce/reduce conflicts

4) Show the actions taken by the parser as it parses "(x)x". Actions should either be of the form "Shift X" or "Reduce R (goto X)" If the parser gets to state 1, it accepts. You can use the following table to help show your work (you may not need all of the rows) (16 points)

Parse stack	Remaining input	Action
0	(x)x	Shift, 3
0 3	x)x	Shift, 2
0 3 2	)x	Reduce 3 (goto 4)
0 3 4	)x	Shift, 5
0 3 4 5	x	Shift, 2
0 3 4 5 2		Reduce 3 (goto 6)
0 3 4 5 6		Reduce 2 (goto 1)
0 1		Accept

2 points per action. Partial credit given (+2 points for each correct action, for partial credit)

Part 6: LR(1) Parsers (ECE 573 only) (20 points):

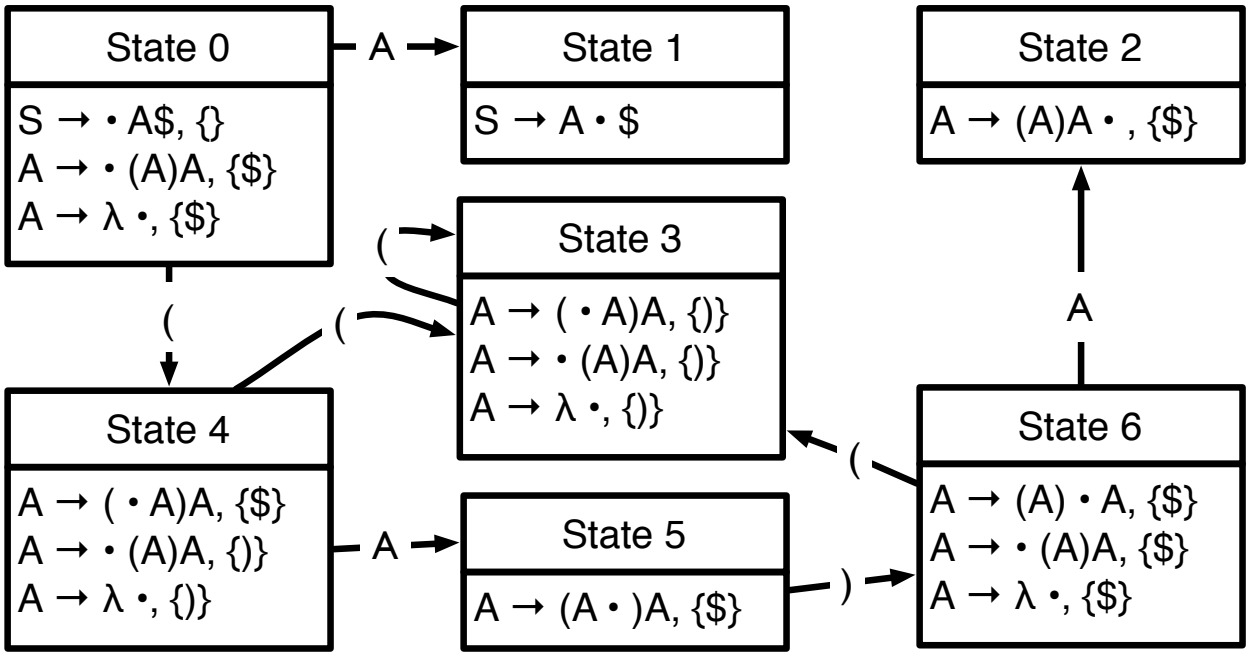
Consider the following slightly modified grammar from Part 5:

$$\begin{aligned} S &\rightarrow A\$ \\ A &\rightarrow (A)A \\ A &\rightarrow \lambda \end{aligned}$$

1) Is this grammar LR(0)? Why or why not? (4 points)

No, because there will be a shift/reduce conflict immediately in State 0.

2) Fill in the missing information from this partial LR(1) machine (you do not have to add any states that are not already shown) (16 points)



2 points per configuration
-1 point per wrong lookahead (partial credit given)