ECE 573 — Midterm 1

September 29, 2009

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	8	
2	15	
3	10	
4	22	
5	25	
6	20	
Total	100	

Part 1: Short answers (8 points)

1) Place the following parts of a compiler in order and identify which are part of the front-end, and which are part of the back-end. *Parser, Code generator, Scanner, Optimizer (of IR), Semantic routines* (2 points)

Scanner, Parser, Semantic routines, Optimizer, Code generator. The first three are the front-end

2) Name two (of four) different types of compilers, and provide a real-world example for each that you name (2 points)

Source-to-source (research compilers); source-to-assembly (gcc, g++, etc.); source-to-bytecode (javac, C# compiler, etc.); bytecode-to-assembly (JIT compilers)

3) Explain (briefly—you shouldn't need more than one sentence) the difference between the *syntax* of a language and its *semantics* (1 point)

The syntax of a language determines its structure, the semantics determine its meaning

4) Explain the difference between a context-free grammar and a context-sensitive grammar (1 point)

Context-free grammar: non-terminals can be replaced by terminals independent of what surrounds them. Context-sensitive grammar: some productions may require that a non-terminal appear in a certain context before they can be applied.

5) Why do semantic routines need to be placed at the end of productions in LR grammars? (2 points)

LR grammars are not predictive; they cannot recognize a production until they see the entire thing. Thus, they cannot apply semantic routines until they've reduced a production, so semantic routines can only go at the end of a production (i.e., at the point where the LR parser will reduce).

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

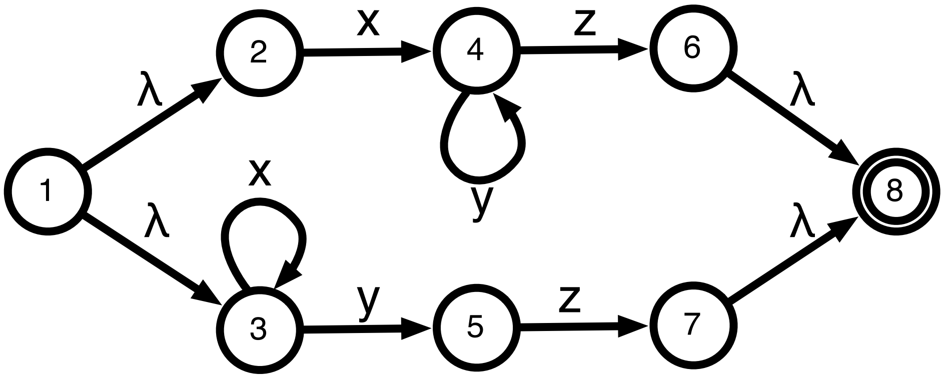
1) Describe, in one sentence, the strings captured by the following regular expression (2 points):

$$(ab)^+c^*$$

One or more instances of “ab” followed by zero or more “c”s

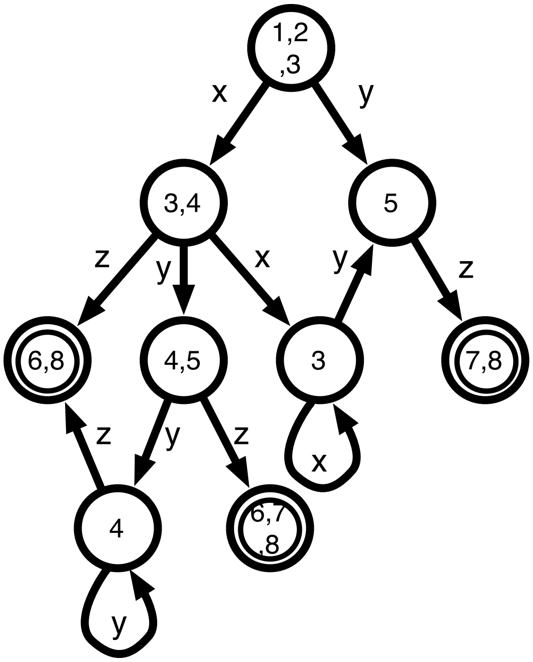
2) Give a finite automaton that accepts languages defined by the following regular expression (this should be a non-deterministic FA) (6 points):

$$((xy^*z)|(x^*yz))$$



3) Give the deterministic equivalent of the NFA you created in problem 2 (7 points)

State	x	y	z	final?
{1, 2, 3}	{3, 4}	{5}		
{3, 4}	{3}	{4, 5}	{6, 8}	
{5}			{7, 8}	
{3}	{3}	{5}		
{4, 5}		{4}	{6, 7, 8}	
{4}		{4}	{6, 8}	
{6, 8}				yes
{6, 7, 8}				yes
{7, 8}				yes



Part 3: Grammars (10 points)

Let G be the grammar:

$$\begin{array}{lcl} S & \rightarrow & AB \\ A & \rightarrow & xB \mid \lambda \\ B & \rightarrow & yA \mid zB \end{array}$$

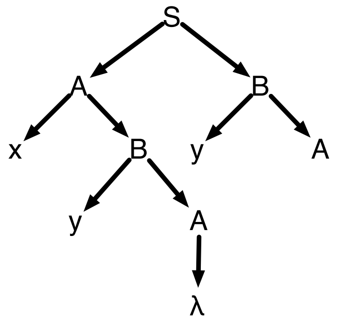
Using this grammar, answer the following questions.

1) What are the terminals and non-terminals of this grammar? (1 point)
Terminals: x, y, z. Non-terminals: S, A, B

2) Give 4 examples of strings in the language defined by this grammar. (2 points)
Many possibilities. Some examples: “y”, “zy”, “xyy”, “xyxzyy”, etc.

3) Draw the parse tree for the following partial derivation (*i.e.*, some of the leaves of your parse tree may be non-terminals) (4 points)

$$S \Rightarrow xyyA$$



4) Did this partial derivation get produced by left-derivation or right-derivation? (1 points)

Left derivation

5) Give an example of a partial derivation produced from S in 2 steps by right derivation (2 points)

Two possibilities:
S ⇒ A B ⇒ AyA

S ⇒ A B ⇒ AzB

Part 4: LL parsers (22 points)

Answer the questions in this part using the following grammar:

$$\begin{array}{lcl} S & \rightarrow & Ab\$ \\ A & \rightarrow & (bAb) \\ A & \rightarrow & (Ab) \\ A & \rightarrow & \lambda \end{array}$$

1) Define the following sets: (8 points)

First(A b *)*	{(, b}
First ((b a b) *)*	{(}
First ((A b) *)*	{(}
Follow (A *)*	{b}

2) Give the predict sets for the productions: (8 points)

Predict(1)	{(, b}
Predict(2)	{(}
Predict(3)	{(}
Predict(4)	{b}

3) Fill in the LL(1) parse table based on your predict sets (4 points)

	(	)	b	$
S	1		1	
A	2, 3		4	

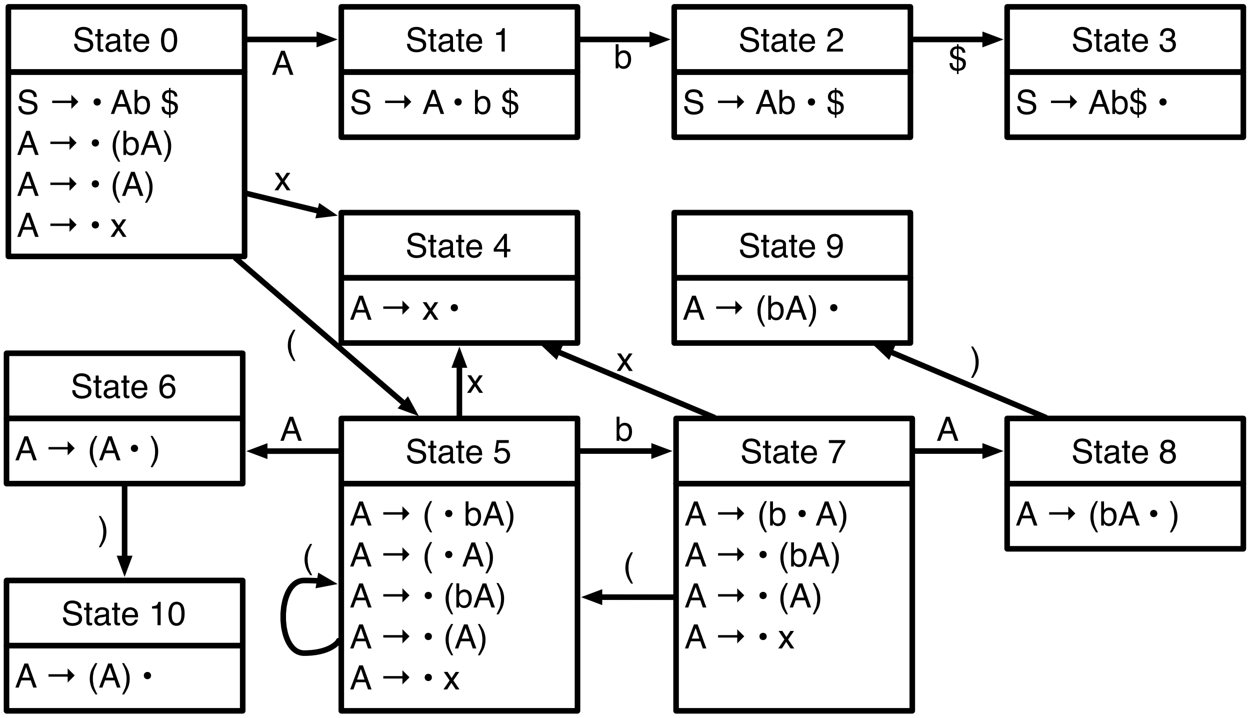
4) Is this an LL(1) grammar? Why or why not? (2 points)
This is not an LL(1) grammar: if we are processing *A* and the lookahead is (, we will not know whether to predict production 2 or production 3.

Part 5: LR(0) Parsers (25 points)

Use the following grammar for the next two questions:

$$\begin{aligned} S &\rightarrow Ab\$ \\ A &\rightarrow (bA) \\ A &\rightarrow (A) \\ A &\rightarrow x \end{aligned}$$

1) Fill in the missing states for the for the following CFSM (12 points) and fill in the missing edge labels (1 point)



2) List the actions the parser will take when parsing the following string. For shift actions, indicate which state the parser will go to; for reduce actions, indicate which rule is being reduced and which state the parser will go to after reducing. (You do not have to show the parse stack or the remaining input—though it may help. Assume the parser accepts when it gets to state 2) (12 points)

(b (x)) b $

S5, S7, S5, S4, R4 (goto 6) [after reducing, parser is in state 5], S10, R3 (goto 8) [after reducing, parser is in state 7], S9, R2 (goto 1) [after reducing, parser is in state 0], S2, Accept

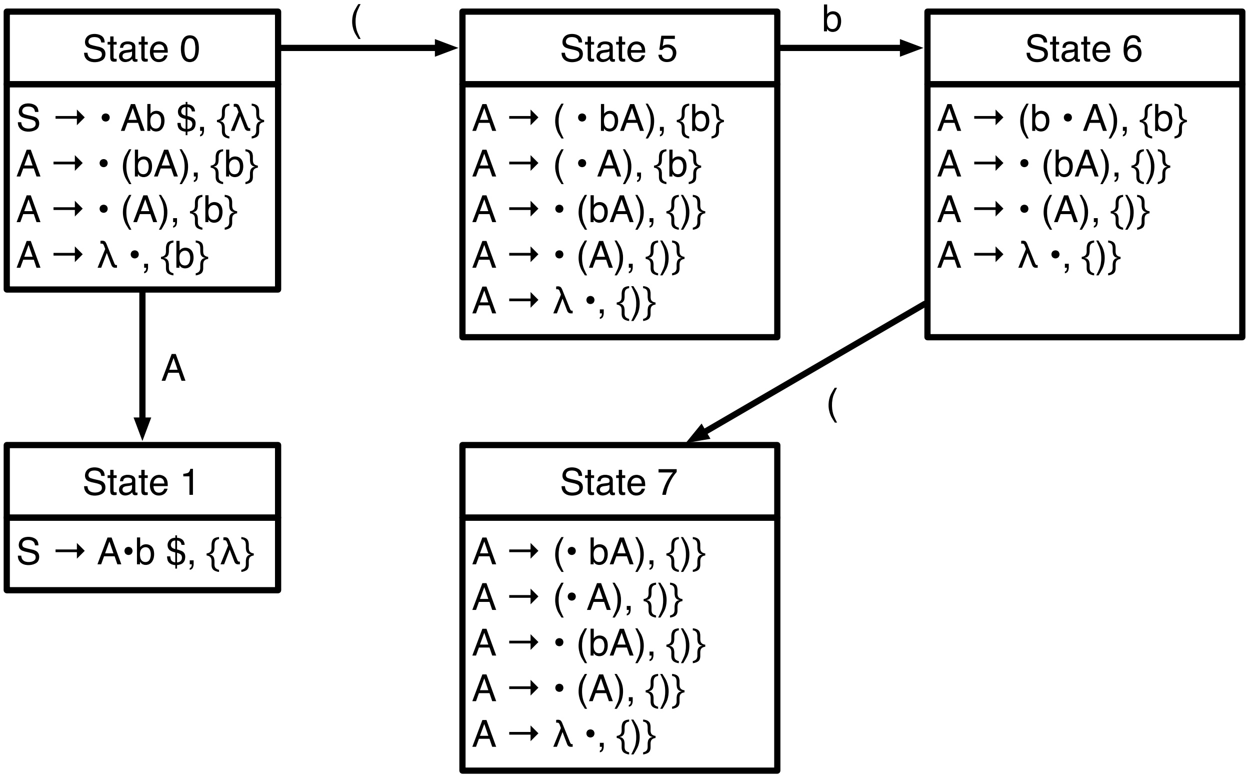
Part 6: LR(1) Parsers (20 points)

Consider the grammar:

$$\begin{array}{ll} S & \rightarrow Ab\$ \\ A & \rightarrow (bA) \\ A & \rightarrow (A) \\ A & \rightarrow \lambda \end{array}$$

1) Is this grammar LR(0)? Why or why not? (1 points)
This is not LR(0). In state 0, we do not know whether to shift or reduce.

2) Fill in the missing states in the partial LR(1) machine given below (13 points)



State 0

S → • Ab $, {λ}
A → • (bA), {b}
A → • (A), {b}
A → λ •, {b}

State 5

A → (• bA), {b}
A → (• A), {b}
A → • (bA), {)}
A → • (A), {)}
A → λ •, {)}

State 6

A → (b • A), {b}
A → • (bA), {)}
A → • (A), {)}
A → λ •, {)}

State 1

S → A•b $, {λ}

State 7

A → (• bA), {)}
A → (• A), {)}
A → • (bA), {)}
A → • (A), {)}
A → λ •, {)}

3) Fill in the action and goto tables for State 0 (4 points)

Action table			
b	$	(	)
R4		Shift	

Goto table					
b	$	(	)	S	A
		5			1

4) How would an SLR parser differ from this LR(1) parser? (1 point) An LALR parser? (1 point)

SLR parser: would use Follow(A) to solve shift/reduce conflict in state 0
LALR parser: would combine states 5 and 7 because they only differ in lookahead