ECE 468/573 — Midterm 1

September 26, 2013

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 _____

Note: ECE 468 students *do not* have to complete Part 6.

Part	Points	Score
1	10	
2	17	
3	12	
4	26	
5	35	
6	15	
Total	115	

Part 1: Short answers (10 points)

1) You are writing a program where different users of the program will use it for very different purposes, and stress different parts of the application. Would you rather compile this program using a regular compiler or a just-in-time compiler? Why or why not? (5 points)

JIT: better able to account for specific behavior of user at runtime when compiling, can optimize for the specific usage scenario.

I was looking for something about “runtime optimization” or “optimization based on usage”

2) Most Android phones, and all iPhones, use processors that support the ARM instruction set. Android phones use software written in a Java-like language, compiled to bytecode, while iPhones use software written in Objective-C, compiled to ARM assembly. Intel developed a line of phones based on their x86-based Atom processor. Explain why its would be easier to have an Atom phone run Android applications than iOS applications (assume that Apple has an x86 port of iOS that Intel can use). (5 points)

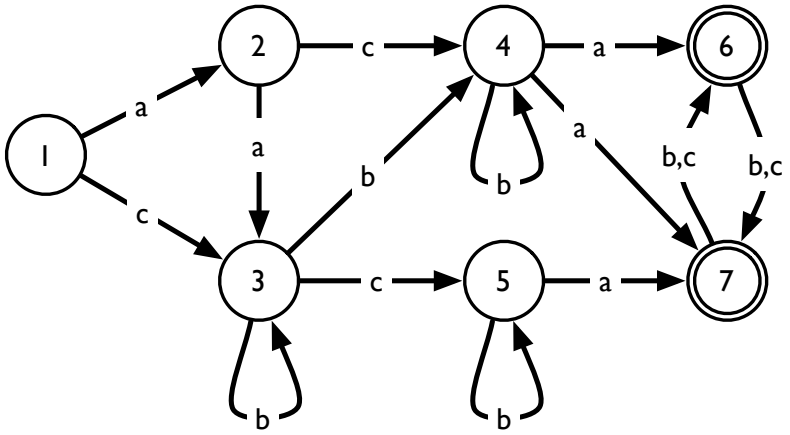
Because Android applications are distributed as machine-independent bytecode. All Intel needs to do is provide a Dalvik VM/JIT, and Android applications can run on Atom. iOS applications would need to be recompiled from source. I was looking for some discussion of machine-independence of Android apps, plus some reference to VMs or JIT compilers.

Part 2: Regular expressions, finite automata and scanners (17 points)

1) Can a language that has strings with some number of 'a's and 'b's (in any order) followed by *at least* as many 'c's as there were 'a's and 'b's be captured with a regular expression? If so, give the regular expression; if not, explain why. (5 points):

No. Need to be able to count the number of 'a's and 'b's, which FAs can't do (and hence no regex possible). Was looking for something about memory or counting

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
1	No	2	—	3
2	No	3	—	4
3	No	—	3, 4	5
4	No	6, 7	4	—
3, 4	No	6, 7	4	5
5	No	7	5	—
6, 7	Yes	—	6, 7	6, 7
7	Yes	—	6	6
6	Yes	—	7	7

3) List which states should be merged when you reduce the above DFA (4 points):

{4} and {5} can be merged. {6}, {7} and {6, 7} can be merged.

Part 3: Grammars (12 points)

Let G be the grammar:

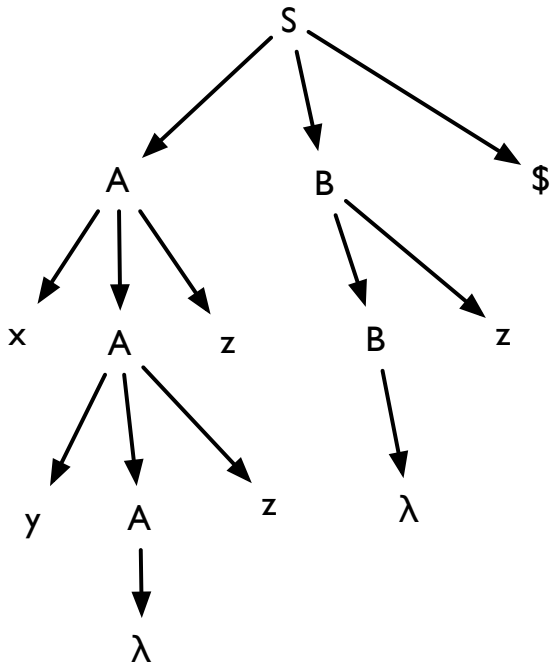
$$
\begin{aligned}
S &\rightarrow AB\$ \\
A &\rightarrow xAz \\
A &\rightarrow yAz \\
A &\rightarrow \lambda \\
B &\rightarrow Bz \\
B &\rightarrow \lambda
\end{aligned}
$$

Using this grammar, answer the following questions.

1) What are the terminals and non-terminals of this grammar? (6 points)

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

2) Draw the parse tree for the string "xyzzz$" (6 points)



Part 4: LL parsers (26 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 points)

First(S) : {x, y, z, $}
First(A): {x, y, λ}
First(B): {z, λ}

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

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

3) Give the Predict sets for each *production* in the grammar (6 points)

Predict(1): {x, y, z, $}
Predict(2): {x}
Predict(3): {y}
Predict(4): {z, $}
Predict(5): {z}
Predict(6): {z, $}

4) Fill in the following parse table (6 points)

	x	y	z	$
S	1	1	1	1
A	2	3	4	4
B			5, 6	6

5) Is the grammar LL(1)? Why or why not? (2 points)

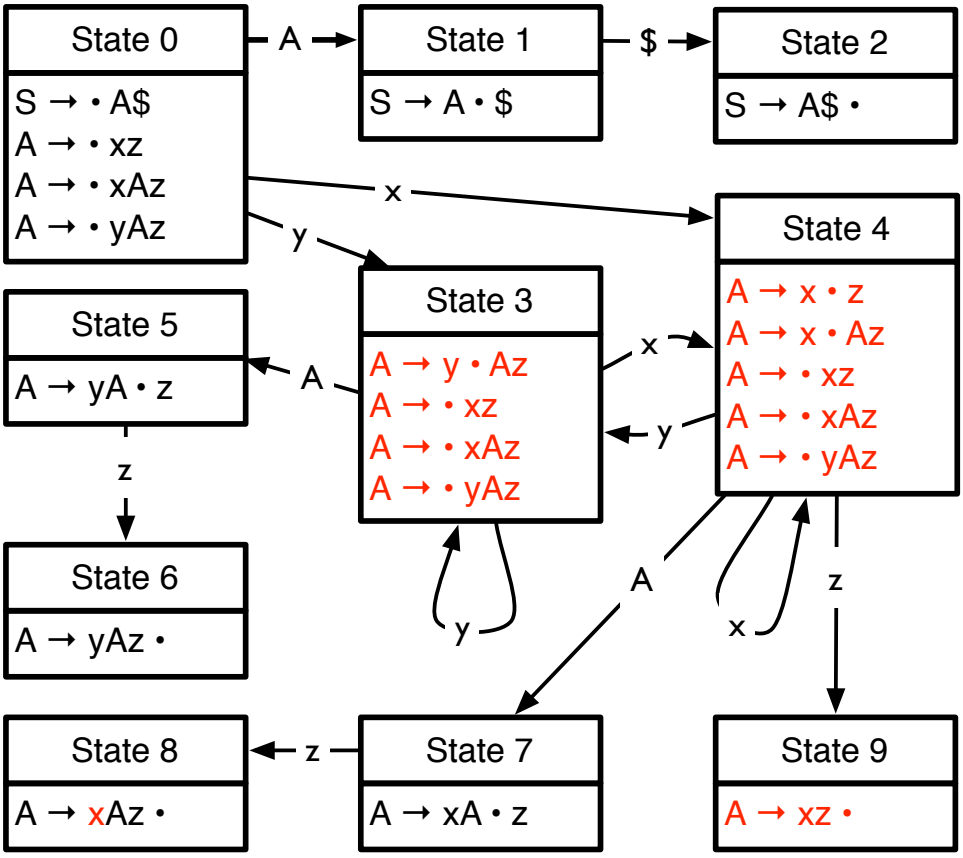
No: predict conflict in the table when expanding B with a lookahead of z.

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

Use the following grammar for the next questions:

$$
\begin{aligned}
1.S &\rightarrow A\$ \\
2.A &\rightarrow xAz \\
3.A &\rightarrow yAz \\
4.A &\rightarrow xz
\end{aligned}
$$

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



2) List the reduce states in the above CFSM, and the shift states (8 points)

Shift: 0, 1, 3, 4, 5, 7
Reduce: 6, 8, 9 (and 2 is an accept state)

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

Yes. No S/R conflicts or R/R conflicts

4) For the following sub-questions, use the CFSM you built in the previous question. Each question will provide the state of the parser in mid-parse, giving the state stack (most recent state on the right) and the next token. For each question, give the action the parser will take next, using the format "Shift X" for shift actions (where X is the state being shifted to) and "Reduce R, goto X" for reduce actions (where R is the rule being reduced, and X is the state the parser winds up in after finishing the reduction). Also provide the new state stack. (3 points each)

a) State stack: 0 3 3 4. Next token: x

Shift [4 0 3 3 4 4]

b) State stack: 0 1 2. Next token: none

Accept

c) State stack: 0 3 4 9. Next token: z

Reduce 4, goto 5 [0 3 5]

d) State stack: 0 4 4 7 8. Next token: z

Reduce 2, goto 7 [0 4 7]

e) State stack: 0 4 3. Next token: z

Parse error (parser will try to shift, but no transition to take when it sees a 'z')

Part 6: LR(1) Parsers (ECE 573 only) (15 points):

1) Consider the grammar from Part 3. Is the grammar LR(0)? Why or why not? (5 points)

No. The lambda rule for A means that State 0 will have a S/R conflict

2) Show the *first* state of the LR(1) machine you would build for the grammar in Part 3 (including lookahead). (5 points)

S → • AB$, {}
A → • xAz, {z, $}
A → • yAz, {z, $}
A → λ •, {z, $}

3) Give the action table entries for the first state (i.e., for each token that could be coming up, say whether the parser would shift or reduce). For reduce actions, say what rule would be reduced. (5 points)

x: shift
y: shift
z: reduce 4
$: reduce 4