The German University Cairo
Faculty of Media Engineering and Technology
Prof. Dr. Carmen Gervet

May 20, 2013

Bar Code

CSIS1003 – Compiler

Final Exam

Instructions: Please Read Carefully Before Proceeding.

1. The allowed time for this exam is **3 hours** (180 minutes).
2. (non-programmable) calculators are allowed.
3. No books or other aids are permitted for this test.
4. This exam booklet contains 17 pages, including this one. Three extra sheets of scratch paper are attached and have to be kept attached. **Note that if one or more pages are missing, you will lose their points. Thus, you must check that your exam booklet is complete**.
5. Please write your solutions in the space provided. If you need more space, **please use the back of the sheet containing the problem or on the three extra sheets and make an arrow indicating that.**
6. When you are told that time is up, please stop working on the test.

All the best.

Please, do not write anything on this page.

Exercise	1	2	3	4	5	6	7	8	9	Total
Maximum Marks	3	5	4	3	5	8	6	3	5	42
Earned Marks										

Exercise 1 Syntax analysis: LL(1) parsing **(3=1+1+1 marks)**

In the following grammars, capital letters are non-terminal and digits are terminals. For each grammar explain why or why not the grammar is LL(1). When it is not LL(1) perform the suitable transformations.

1.1

S → 0 | 1 | 23

Answer: It is LL(1), deterministic choice of production for a given input.

1.2

S → 0 |T 1

T→ 1 | S 0

Answer: It is not LL(1), left recursive S -> T1 -> S01
Eliminate left recursion:
S -> 0 | T1
T -> 1 | S0 substitute S in T -> 1 | S0 to obtain T -> 1 | 00 | T10
Eliminate immediate left recursion on latest production leading to:
S -> 0 | T1
T-> 00 B | 1 B
B -> 10 B | ε
Or the other way round: S -> 0 T | 11 T
T -> 01 T | ε

1.3

S → 0 | 11 | 01

Answer: It is not LL(1), needs left-factoring.
S -> 0T | 11
T -> 1 | ε

Exercise 2. FIRST and FOLLOW sets **(5=2+1+1+1 marks)**

Consider the following grammar.

S -> A a
S -> a
A -> c
A-> b A

2.1 Compute FIRST and FOLLOW sets for S and A.

Answer:
FIRST(S) = {c,b,a}, FIRST(A) = {c,b},
FOLLOW(S) = {$}, FOLLOW(A) = {a}

2.2. Construct an LL(1) parsing table for this grammar.

	a	**b**	**c**	**$**
S				
A				

Answer:

S	S-> a	S-> Aa	S-> Aa	
A		A-> bA	A-> c	

2.3 Are the FOLLOW sets necessary for constructing this table? Why or why not?

Answer: No they are not because they are only used, if we have ε in a FIRST set.

2.4 Show the parse tree for parsing the input string: bbca

Answer:

S
A **a**
b **A**
b A
c

Exercise 3 Syntax analysis: bottom-up parsing **(4=2+2 marks)**

We call a simple grammar, one that has a maximum of 3 non-terminals and 4 productions

NOTE: S →P | Q represents 2 productions.

3.1 Give an example of a simple grammar with a shift-reduce conflict.

Answer:
One example: S -> Ta | a
T -> ε
Another simple example: S -> S | ε

3.2 Give an example of a simple grammar with a reduce-reduce conflict.

One answer:
S -> A | B
A -> a
B -> a
A common error in this problem was to forget a starting rule for the grammar and just giving
A -> a
B -> a which can not work
Or giving S -> AB which will not generate a reduce/reduce conflict.

Exercise 4 Constructing LR tables (3 marks)

In the context of the construction of an action table for bottom-up parsing, a student produced the table below corresponding to a grammar whose rules are not shown and whose terminal and non-terminal symbol sets are respectively {a, x, +} and {S, I}.

Which of the following statements is true according to the material studied this term?

	a	X	+	$	S	I
0		S3			1	
1	S6		S2	acc		
2		S4				5
3	R1		R1/S2	R1		
4	S1			S5		
5	R2			R2	2	
6			S1			6

a) **The parsing table could not have been constructed using the LR(0) algorithm because it has conflicts**

b) **The parsing table could not have been constructed using the SLR(1) algorithm because it has conflicts**

c) **The parsing table could not have been constructed using any LR technique because it is wrong**

d) **All of the above are correct**

Justify your choice.

Answer: c) because we cannot have a shift on a $ input. $ can only be the end of the input string. So it is either accept or a reduce.

MARKING for this question:

- 1.5 points if the student figured out that c) was correct because the other ones were wrong but could not find the reason.
- 2 points if mentions no shift on $ even though another answer was given like d).
- 0 otherwise

Exercise 5 Shift-reduce parsing **(5=1+4 marks)**

Given the grammar below

1. **S $\rightarrow$ ABCD**
2. **D $\rightarrow$ d**
3. **C$\rightarrow$ cc**
4. **B$\rightarrow$ Bb**
5. **B$\rightarrow$b**
6. **A $\rightarrow$ gBa**

5.1 Indicate the handle in the following right-sentential form gbbabbccd.

Answer: the handle is underlined and in bold. g**b**babbccd

It corresponds to the first substring to be reduced in a shift-reduce parsing.

5.2 The grammar is now extended with the semantic actions as follows. The variable x is global, i.e. all the semantic actions update the same x. What is the final value of x after running a shift-reduce parse on the above input? x is first initialized to 0. Show your reasoning.

1. S $\rightarrow$ ABCD	**{ x = 2x + 2; }**
2. D $\rightarrow$ d	**{ x = x + 4; }**
3. C$\rightarrow$ cc	**{ x = 2x + 6; }**
4. B$\rightarrow$ Bb	**{ x = 3x +1; }**
5. B$\rightarrow$b	**{ x = x +1; }**
6. A $\rightarrow$ gBa	**{x = 5x;}**

Answer: Sequence of production used in a shift reduce parser, together with the value of x

5. B-> b	{x=1}	g**B**babbccd
4. B-> Bb	{x=4}	g**B**abbccd
6. A-> gBa	{x=20}	**A**bbccd
5. B-> b	{x=21}	A**B**bccd
4. B-> Bb	{x=64}	A**B**ccd
3. C-> cc	{x=134}	AB**C**d
2. D-> d	{x=138}	ABC**D**
1. S-> ABCD	{x=278}	S

NOTE: most errors corresponded to parsing top town, and thus computing x in the wrong order.

Exercise 6 SLR(1) parsing **(8=3+1+2+2 marks)**

Consider the following grammar for arithmetic operations in postfix notation.

$$S \rightarrow S\ S\ +$$
$$S \rightarrow S\ S\ *$$
$$S \rightarrow a$$

6.1 It is not suitable for top-down parsing and we want to investigate whether it is suitable for LR(0) and SLR(1) parsing.
Build the LR(0) DFA for the augmented grammar.

6.2 Is it LR(0)? Justify your answer.

6.3 Determine whether the grammar is SLR(1):

a) Compute the FIRST and FOLLOW sets for the non terminal 'S'

b) Is it SLR(1)? Build the SLR(1) parsing table.

Exercise 7 Semantic analysis - Dependency graph **(6=2+1+2+1 marks)**

Consider the following attribute grammar with semantic rules:

Grammar rule	Semantic rules
S → A B C	**B.u = S.u** **S.v = B.v +C.v** **A.u = S.v**
A → int	**A.v = 2 * A.u**
B → id	**B.v = B.u**
C → 0	**C.v = 1**

7.1 Draw the parse tree for the input 'int id 0' (the only string for the language), and show the dependency graph for the associated attributes. Note that inherited attributes are conventionally written to the left of the node while synthesized attributes are posted to the right.

7.2 Is this grammar L-attributed, S-attributed or neither? Use the term that best describe the grammar.

7.3 Assume that S.u is assigned the value 3 before starting attribute evaluation, what will be the value of A.v when evaluation has terminated?

7.4 Suppose now that the semantic rules were instead:

Grammar rule	Semantic rules
S → A B C	**B.u = S.u** **S.v = B.v +C.v** **A.u = S.v** **C.u = S.v**
A → int	**A.v = 2 * A.u**
B → id	**B.v = B.u**
C → 0	**C.v = C.u – 2**

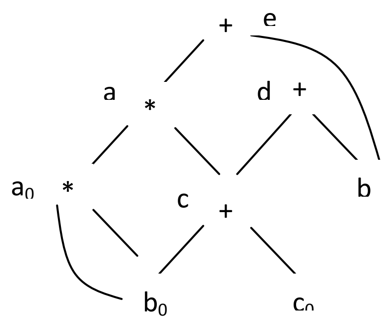
What value does A.v have after evaluation of all attributes, if the initial value of S.u is 3?

Explain why this result occurred?

Exercise 8 DAG and Three-address code **(3=2+1 marks)**

The three-address code is a linearized version of the DAG.

8.1 Given a Directed Acyclic Graph, generate the corresponding three-address code.



8.2 If only 'd' is live on exit, how can you simplify the DAG. Show the resulting DAG.

Exercise 9 DAG, code optimization **(5=2+2+1 marks)**

Consider the following basic block, in which all variables are integers, and ^ denotes exponentiation.

a = b + c
z = a ^ 2
x = 0 * b
y = b + c
w = y * y
u = x + 3
v = u + w

9.1 Derive the directed-acyclic graph for this basic block

9.2 Assume that the only variables that are live on exit of this block are ‘v’ and ‘z’. Apply the following sequence of optimization steps once (follow the imposed order):
1) Algebraic simplification
2) Common sub-expression elimination,
3) Constant folding
4) dead code elimination
Enter the results in the table below:

<table>
<tr><td>Original code:

a = b + c
z = a ^ 2
x = 0 * b
y = b + c
w = y * y
u = x + 3
v = u + w</td><td></td></tr>
<tr><td>1) Result of algebraic simplification</td><td>3) Result of constant folding</td></tr>
<tr><td>2) Result of common sub-expression elimination</td><td>4) Result of dead code elimination</td></tr>
</table>

9.3 If now the live variables are ‘z’,’x’, and ‘a’ show how the initial DAG can be simplified.

Extra sheet 1

Extra sheet 2

Extra sheet 3