The German University Cairo **May 26, 2010**
Faculty of Media Engineering and Technology
Prof. Dr. Carmen Gervet

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 14 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**	**Total**
Maximum Marks	6	5	5	5	12	6	8	11	58
Earned Marks									

Exercise 1 Ambiguous grammars (6 = 3+3 marks)

Consider the following grammar, which generates all the non-empty strings of balanced parenthesis.

P → (P) | PP | ( )

1.1 Prove that this grammar is ambiguous

Answer: to prove ambiguity one needs to show that we can derive 2 distinct parse trees for a given input string. Consider the string: ( ) ( )( )

```
         P                                                    P

P        P                                           P        P

( )      P        P                         P        P        ( )

( )      ( )      ( )                       ( )      ( )
```

1.2Modify the grammar so to obtain a non-ambiguous grammar generating the same language

Answer: An approach to remove ambiguity is to turn the left recursion into a right recursion thus P is either call itself (P) or call a terminal ( ) with Q afterwards.

A non ambiguous grammar:

P -> (P)

P -> ( ) Q

Q -> P Q | ε

NOTE: many answer lead to grammars that could include the empty string which is not part of the initial language.

Exercise 2 Shift-reduce parsing **(5=1+4 marks)**

Given the grammar below

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

i) 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.

ii) 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 → ABCD**	**{ x = 2x + 2; }**
2.	**D → d**	**{ x = x + 4; }**
3.	**C→ cc**	**{ x = 2x + 6; }**
4.	**B→ Bb**	**{ x = 3x +1; }**
5.	**B→b**	**{ x = x +1; }**
6.	**A → 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 3 bottom-up parsing , SLR(1) (5 marks)

Prove that there is a grammar with a <u>single production</u> that has a shift-reduce conflict under SLR(1) parsing rules. In other words provide a grammar with 1 production only and show that it is not SLR(1).

An answer: S -> SaS

The main confusion lied in the concept of a shift-reduce conflict and whether the grammar needs to produce a language or not.

conflict state {S-> Sa.S, S-> .SaS} yields on goto on S to state {S-> SaS., S.aS} this state has a s/r conflict that is not resolved with SLR(1) because ‘a’ belongs to follow (S). Thus we can reduce using the production or shift on ‘a’.

This exercise lead to quite few errors and confusions with respect to the detection of SLR(1) properties. The main error was the confusion between LR(0) and SLR(1).

The production S-> ε yields the state: S-> . which has a conflict if we consider $ as symbol to shift upon. However, as we saw in the mock exam, we do not shift on $ we ‘accept’ (or reduce). Thus the $ column in the SLR(1) table contains only reduce or acc actions.

Also on non-terminals we do not shift but ‘goto” another state of the DFA(0). Thus productions of the form S-> SS does yield the state

I0 ={S’->.S , S-> .SS}

I1= {S’-> S., S-> S.S, S-> .SS}

I2 = {S-> SS. S->S.S S->.SS}

The SLR parsing table for this production is, given the follow set of S={$}

State	Action	GOTO
	$	S
I0		1
I1	acc	2
I2	r1	2

Only terminals are involved in shift moves.

Exercise 4 **(5 marks)**

Give a single production grammar that is SLR(1) but not LL(1). Justify your answer.

Answer:

S -> Sa

Not LL(1) because left-recursive. There is no first set for S!

The DFA(0) states are:

I0 = { S' -> .S, S -> .Sa}

I1 = { S'->S. , S-> S.a}

I2 = { S-> Sa.}

It is SLR(1) because there is no conflict in the SLR(1) parse table. FOLLOW(S) = {a,$}

State	Action		GOTO
	a	$	S
I0			1
I1	S2	acc	
I2	r1		

Exercise 5 Constructing SLR(1) parsing (12=4+2+3+3 marks)

Consider the "dangling-else" grammar:

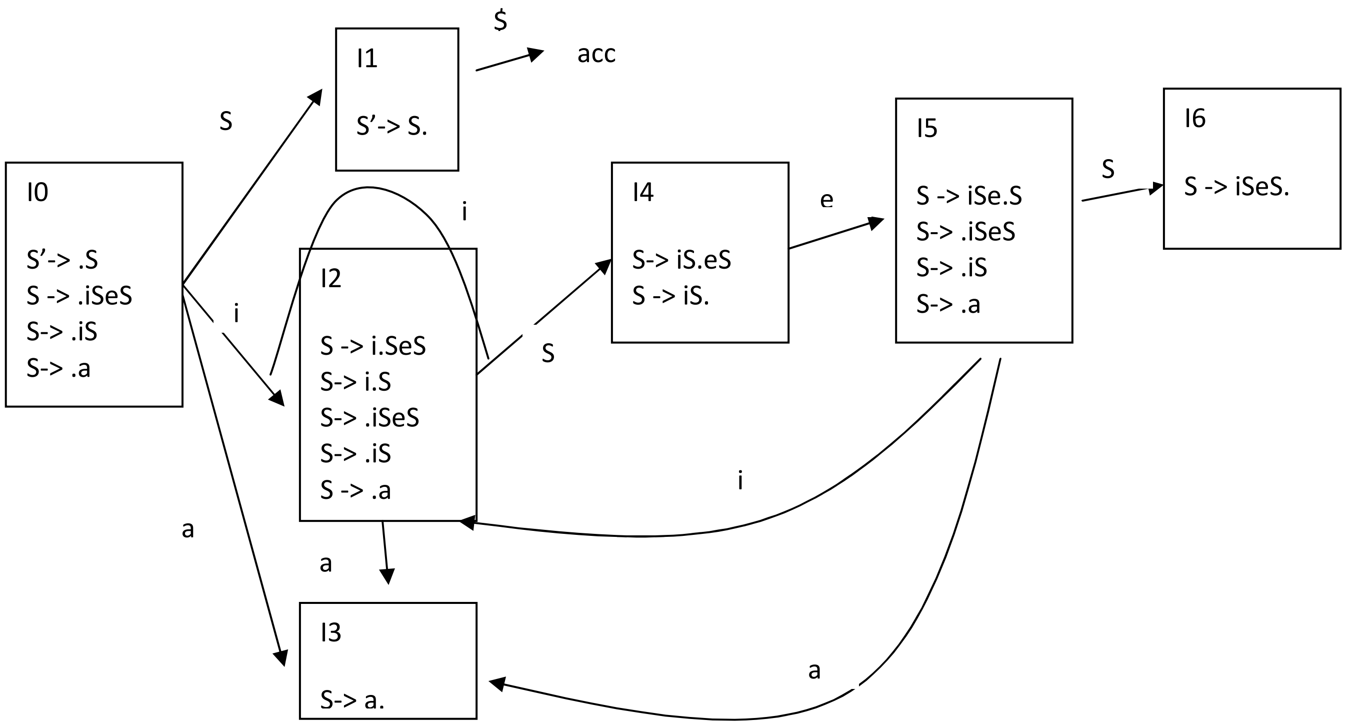
S -> if-e-then S else S | if-e-then S | other

simplified in its syntax by the grammar:

S -> i S e S | i S | a

This grammar is ambiguous and not suitable for top-down parsing.

1. **Build the LR(0) DFA for the augmented grammar (using the simplified notation).**



2. **Is the grammar LR(0)? Justify your answer.**

Answer: No it is not LR(0) because there is a shift reduce in state 4.

3. **Now a student claims that this grammar is SLR(1). Is the student correct?**

Justify your answer by computing the follow set of S and constructing the SLR(1) parsing table.

FOLLOW(S) ={e,$}

STATE	ACTION				GOTO
	i	e	a	$	S
I0	S2		S3		1
I1				Acc	
I2	S2		S3		4
I3		R3		R3	
I4		S5/R2		R2	
I5	S2		S3		6
I6		R1		R1	

The main mistakes came from the absence of using the follow set to reduce on a production. And the consequences of an incomplete DFA(0).

4. What will be the parsing actions on input "iiaea"?

Line	Stack	Symbols	Input	Action
1)	0	$	iiaea$	Shift 2
	0 2	$i	iaea$	Shift 2
	0 2 2	$ii	aea$	Shift 3
	0 2 2 3	$iia	ea$	Reduce 3
	0 2 2 4	$iiS	ea$	Conflict!!

Exercise 6 Semantic analysis – attribute grammars **(6=2+4 marks)**

Consider the following grammar, that represents numbers in the form of integers (e.g. 12), reals (e.g 3.14), and numbers in exponent format (e.g 12.4E5).

^ denotes "to the power of"

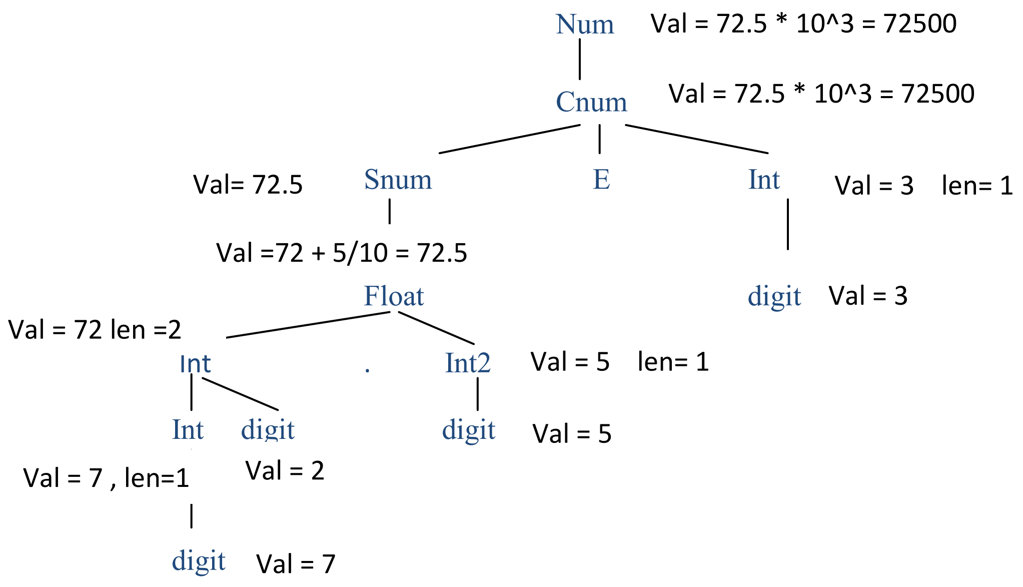
Num → Snum	**{Num.val = Snum.val}**
Num → Cnum	**{Num.val = Cnum.val}**
Snum → Int	**{Snum.val = Int.val}**
Snum → Float	**{Snum.val = Float.val}**
Float → Int1 '.' Int2	**{Float.val = Int1. Val + Int2.val / 10^Int2.len }**
Cnum→ Snum 'E' Int	**{Cnum.val = Snum.val* 10^Int.val}**
Int → digit	**{Int.val = digit.val; Int.len =1}**
Int → Int1 digit	**{Int.val = Int1.val* 10 + digit.val; Int.len = Int1.len + 1}**

6.1 Is this grammar L-attributed, S-attributed or neither? Justify your answer.

Answer: all attributes are synthesized thus it is S-attributed.
It is also L-attributed since it is a superset of S-attributed but the main point is about the nature of the attributes in the semantic rules. The reason for the type of grammar must be given.

6.2 Draw the parse tree for the input string 72.5E3
Draw the dependency graph over the parse tree and indicate the valuation of the attributes.

Answer:



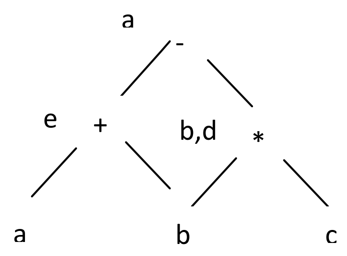
To compute the attribute values, go through each production as it is used to build the parse tree bottom-up and run the semantic rules. They determine the value of each attribute.

Exercise 7 DAG and Three-address code **(8=2+3+3 marks)**

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

7.1 Construct the DAG for the basic block

```
d = b * c
e = a  + b
b = b * c
a = e – d
```



7.2 Simplify the three-address code (using the local optimization techniques you know!) assuming

a) Only *a* is live on exit from the block.

Answer:

d = b * c
e = a + b
~~b = b * c~~
a = e – d

b) *a,b,* and *c* are live on exit from the block.

Model answer:

~~d = b * c~~
e = a + b
b = b * c
a = e – b

Exercise 8 Code optimization **(11=2+3+1+3+2 marks)**

8.1 Given the following code, what are the sequences of quadruples that we can apply algebraic identities code optimization techniques to (including arithmetic identities and constant folding)? Just identify the sequences.

1. **(=,10, ,b)**
2. **(=, 0, ,i)**
3. **(*, c,i, t1)** <- t1=0
4. **(*, c, 1, t4)** <- t4 = c
5. **(<, i, 10, t0)**
6. **(jfalse,t0, , 14)**
7. **(+,b, t1, t2)** <- t2 = b
8. **(*, t2, d, t3)** <- t3 = b*d
9. **(*, t4, d, t5)** <- t5 = c*d
10. **(= t3, , a)**
11. **(+,i , 1, i)**
12. **(+, t3, t5, t3)**
13. **(jmp, 5, ,)**
14. **(+, b, 1, b)** <- b = 11

8.2 Apply "algebraic identities" optimization combined with constant propagation to the sequence of quadruples in 9.1 producing an optimized set of quadruples.

Answer:

1. (=,10, ,b) /** left or removed, both are accepted since b is not used throughout and is given a new value on Exit **/
2. (=, 0, ,i)
3. ~~(=, 0, ,t1)~~
4. ~~(=, c, , t4)~~
5. (<, i, 10, t0)
6. (jfalse,t0, , 14)
7. ~~(=,10, ,t2)~~ /** t2 is never used
8. (*, 10, d, t3)
9. (*, c, d, t5)
10. (= t3, , a) /** a must be kept because no indication whether live or not on exit
11. (+,i , 1, i) /** we have a conditional loop thus 'i' must be kept.
12. (+, t3, t5, t3)
13. (jmp, 5, ,)
14. (=, 11, , b)

8.3 How many temporaries are left in the optimized code?

Answer 3 (t0, t3, t5)

8.4 Derive the basic blocks for the optimized code

Answer:

```
B1: b = 10
    i = 0

B2:   t0: (i<10)
     (jfalse, t0, , B4)

B3: t3 = 10*d
    t5 = c*d
    a= t3
    i = i+1
    t3 = t3+t5
    goto B2

B4: b = 11
```

8.5 Construct the flow graph over the basic blocks derived in 9.4

Answer:

$$\begin{tikzcd}
B1 \arrow[r] & B2 \arrow[r] \arrow[d, bend left] & B4 \\
& B3 \arrow[u, bend left] &
\end{tikzcd}$$