Key Solution

King Saud University
College of Computer and Information Sciences
Computer Science Department

Final Exam
Academic Year: 2015/2016
First Semester

BSc Program
Course Name/No. : Programming Language Compilation / CS340

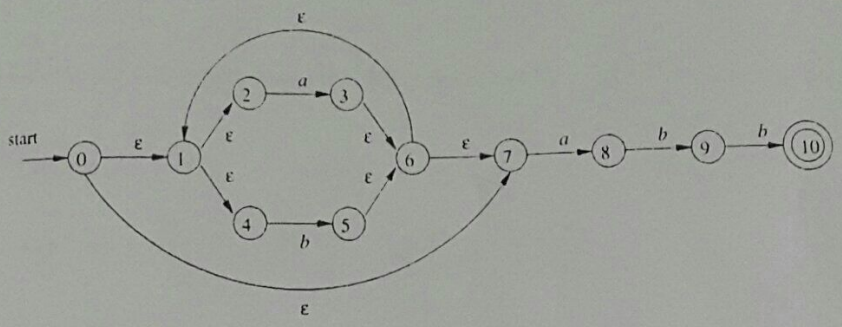
Exam Date: 30 **/12/2015**:

Total Number of Pages: 8 pages (including this cover page)

Student Name	
Student ID.	

Question No.	Total Mark	Student Mark
1.	**6**	
2.	**7**	
3.	**8**	
4.	**8**	
5.	**4**	
6.	**7**	
Total	**40**	

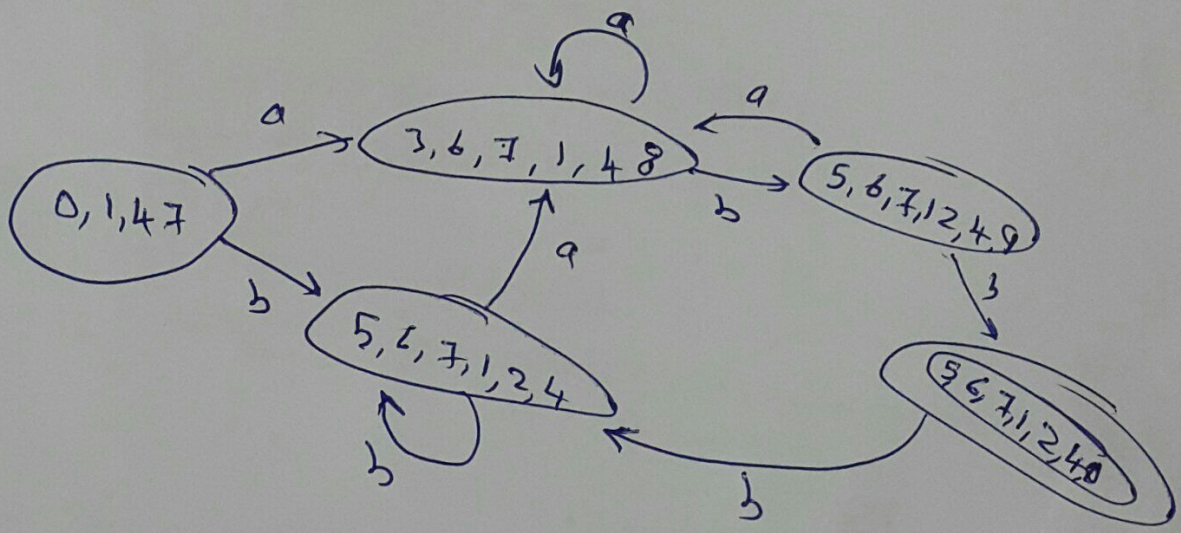
Q1) Consider the following NFA



A) Describe the language of this NFA using a regular expression. (2 marks)

$(a+b)^* abb$

B) Convert the above NFA into a DFA. (4 marks)



Q2) Consider the CFG

E → E-E | E+E
E → int | id

a) Rewrite the CFG in such a way that allows you to write a recursive descent parser for it. (2 marks)

Eliminate left recursion

$E \rightarrow intE' \mid idE' \mid int \mid id$

$E' \rightarrow +E \mid -E$

b) Write a recursive descent parser in Java (or as s pseudo code) for the grammar you wrote in above (in A). (3 marks)

```
bool term(Token tok){
        return next++tok;
}
bool E1(){return term(int) && Ep();}
bool E2(){return term(id) && Ep();}
bool E3(){return term(id);}
bool E4(){return term(int);}
bool E(){Token save=next; return(next=save, E1()
                || next=save, E2()
                || next=save, E3()
                || next=save, E4());}
bool Ep1(){return term(+) && E();}
bool Ep2(){return term(-) && E();}
bool Ep(){Token save=next; return(next=save, Ep1() ||
                (next=save, Ep2());}
```

c) Rewrite the grammar in such a way that makes an LL(1) parser possible for the grammar. (2 marks)

$E \rightarrow intX \mid idX$

$X \rightarrow E' \mid \varepsilon$

$E' \rightarrow +E \mid -E$

Q3) A) What is the main reason that makes parsers unable to detect all errors? Give examples of 4 errors that cannot be detected by a parser. (3 marks)

Reason:

because CFG cannot or discribe capture all prog. lang. constructs

Examples: 1)

using undeclared variable

2)

data type mismatch

3)

using a void function in expression

4) A class that does not correctly implement an interface.

b) In your own words, describe what each of the following rules mean. (2+1 marks)

1.

$$\frac{\begin{array}{l} f \text{ is an identifier.} \\ f \text{ is a non-member function in scope S.} \\ f \text{ has type } (T_1, \ldots, T_n) \to U \\ S \vdash e_i : T_i \text{ for } 1 \le i \le n \end{array}}{S \vdash f(e_1, \ldots, e_n) : U}$$

if each of the arguments $e_1, e_2, \ldots, e_n$ has type $T_1, T_2, \ldots T_n$, then the return type of f is U

2.

$$\frac{\begin{array}{l} S \vdash e_1 : T \\ S \vdash e_2 : T \end{array}}{S \vdash e_1 = e_2 : T}$$

if e_1 has type T and e_2 also has type T, then $e_1 = e_2$ has type T

B) Assuming dynamic type checking, rewrite the above rules so that they work for reference types. (2 marks)

1.

f is an id
f is a function in scope S
f has type $(T_1, \ldots, T_n) \to U$
$S \vdash e_i : R_i$ and $R_i \leq T_i$ for $1 \leq i \leq n$

$S \vdash f(e_1, \ldots, e_n) : U$

2.

$S \vdash e_1 : T_1$
$S \rightarrow e_2 : T_2$
$T_2 \leq T_1$

$S \vdash e_1 = e_2 : T_2$

Q4 Consider the following DFA that recognizes the viable prefixes for the grammar

E→ E
E→ E+T | T
T→T*F | F
F→(E) | id

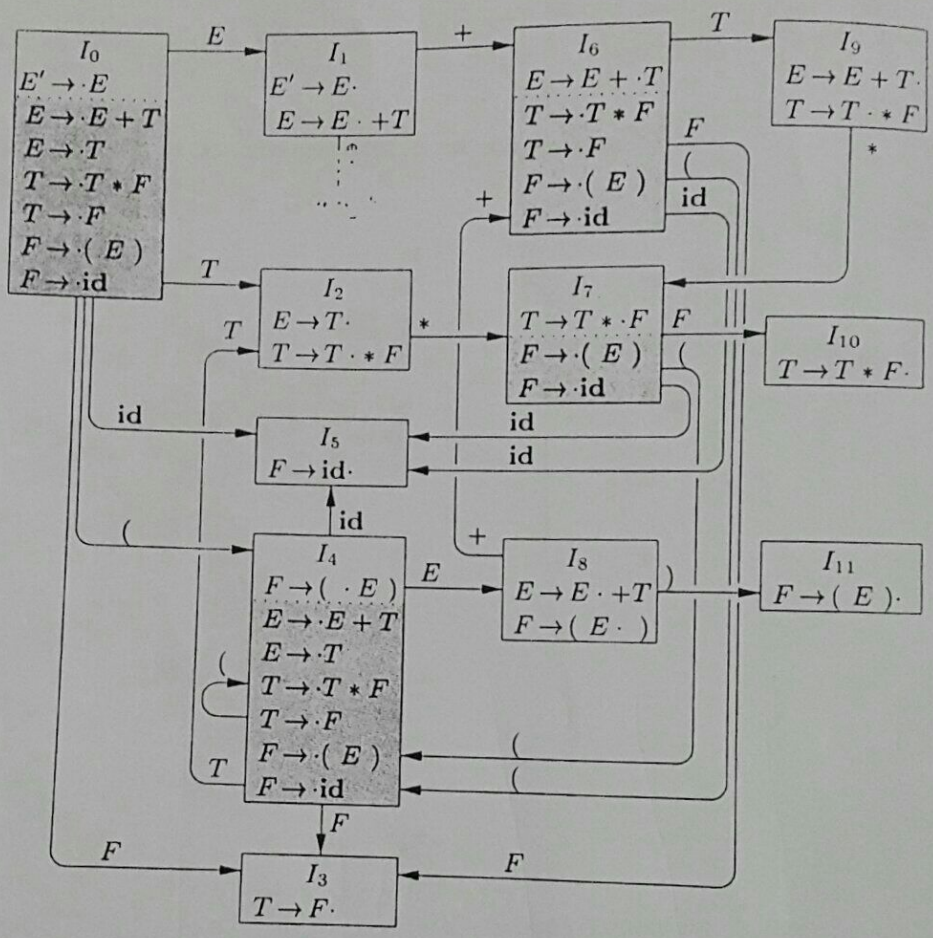
A) Construct an LL(1) table for the grammar. (4 marks)

	+	*	(	)	id	$
E'			E		E	
E			T/E+T		T/E+T	
T			F/T*F		F/T*F	
F			(E)		id	

B) Is the grammar an LL(1) grammar? Why or why not? (1 mark)

No, because it has a multiply defined entry.

c) The following NFA recognizes the viable prefixes for the above grammar. Is the grammar an SLR(1) grammar? Why or why not? (3 marks)



Yes it is because + is not in Follow of E' and * is not in the follow of E

Q5) Write cgen for each of the following expressions (Hint you can use the MIPS instructions at the last page). (3+1 marks)

1) If e1=e1 then e3 else e4

```
cgen (if e1=e2 then e3 else e4)
    cgen(e1)
    sw $a0 0($sp)
    addiu $sp $sp -4
    cgen(e2)
    lw $t1 4($sp)
    addiu $sp $sp 4
    beq $a0 $t1 true_if
    cgen(e4)
    b endif
true_if:
    cgen(e3)
end_if:
```

2) 5

```
li $a0 5
```

Q6) A) Write code that cgen generates for following procedure

def add(x,y)=x+y+2;

```
f_entry:
    move $fp $sp
    sw $ra 0($sp)
    addiu $sp $sp -4
    lw $a0 4($fp)
    sw $a0 4($sp)
    addiu $sp $sp -4
    lw $a0 8($fp)
    lw $t1 4($sp)
    addiu $sp $sp 4
    add $a0 $t1 $a0
    lw $ra 4($sp)
    addiu $sp $sp 4
    lw $fp 0($sp)
    jr $ra
```

B) Write the code that cgen generates for the invocation statement

add(4,5)

```
    sw $fp 0($sp)
    addiu $sp $sp -4
    li $a0 4
    sw $a0 0($sp)
    addiu $sp $sp -4
    li $a0 5
    sw $a0 0($sp)
    addiu $sp $sp -4
    jal f_entry
```