MODEL MIDSEMESTER SOLUTION

Jot Sarup Singh (11CS10059)

1)

a) A Java source program may first be compiled into an intermediate form called *bytecodes*. The bytecodes are then interpreted by a virtual machine. The virtual machine ensures platform independence.

b) Both a.>b and b.>a will hold in a grammar where there are operators with same precedence and the grammar is left associative. For example, let a = '+' and b = '-'.

c) E-closure(X):

```
Push all states of X onto stack;
Initialize E-closure(X) to X;
while (stack is not empty){
        pop t, the top element off the stack;
        for ( each state u with an edge from t to u labeled E )
                if ( u is not in E-closure(X)) {
                        add u to E-closure(X);
                        push  u onto stack;
                }
}
```

d) Consider the state:
S -> L.=R
R -> L.
and suppose '=' is in FOLLOW(R).
Then, the first item in the set makes ACTION[2,=] be shift.
and the second item makes ACTION[2,=] be reduce.

e) Letter -> (a-zA-Z)
Digit -> (0-9)
Email -> (letter)(letter|digit)*(@)(letter+)(.)(letter+)

2)

Stmt -> **if** Exp **then** Stmt **else** Stmt | **if** Exp **then** Stmt | a

Exp -> b

(i) Perform Left factoring on the above grammer.

(ii) Design a predictive parsing table.

(iii) comment on that generated parsing table.

Answer :

(i) After Left factoring

Stmt -> **if** Exp **then** Stmt S' | a

S' -> **else** Stmt | epsilon

Exp -> b

(ii) **First Set**

if	if
then	then
a	a
else	else
epsilon	epsilon
b	b

Follow Set

Stmt	$, else, epsilon
S'	$, else, epsilon
Exp	then

	If	Then	Else	A	B	$
Stmt	Stmt ->**if** Exp **then** Stmt S'			Stmt -> a		
S'			S' ->**else** Stmt S' -> epsilon			S' -> epsilon
Exp					Exp -> b	

(iii) The above predictive parsing tabel has a conflict in the cell (S' , Else) and hence is not LL(1) Grammer.

(b) There will be a conflict because the following rule has not been added:

If Beta -> Epsilon then alpha does not derive any string beginning with terminal in FOLLOW (A)

3) a) These grammers have the property (among other essentail requirements) that no production right side is epsilon or has the adjacent non terminals.

The techinque is a manipulation of tokens without the knowledge of the underlaying grammer. In fact once we may effectively ignore the grammer, using the non terminals on th stack anly as placeholders for attributes associated with the non terminals.

b)

E -> E op E | id

op - > + | - | * | /

<u>Equivalent operator grammer</u>

E -> E + E | E – E| E * E | E / E | id

	+	-	*	/	Id	$
f	2	2	4	4	4	0
g	1	1	3	3	5	0

Operator Precedence Table

	Id	+	-	*	/	$
id		>	>	>	>	>
+	<	>	>	<	<	>
-	<	>	>	<	<	>
*	<	>	>	>	>	>
/	<	>	>	>	>	>
$	<	<	<	<	<	

b) Parse P * Q / R + T

Stack	Input
$	P * Q / R + T $
$ P	* Q / R + T $
$ id	* Q / R + T $
$ id *	Q / R + T $
$ id * Q	/ R + T $
$ id * id	/ R + T $
$ id	/ R + T $
$ id /	R + T $
$ id / R	+ T $
$ id / id	+ T $
$ id	+ T $

$ id +	T $
$ id + T	$
$ id + id	$
$ id	$
$	$

4) 1. S' -> S
2. S -> aABe
3. A -> Abc
4. A -> b
5. B -> d

	FIRST	FOLLOW
S'	a	$
S	a	$
A	b	b,d
B	d	e

	a	b	c	d	e	$	S	A	B
0	s2						1		
1						accept			
2		s4						3	
3		s6		s7					5
4		r3		r3					
5					s9				
6		s8							
7					r4				
8		r2		r2					

9 r1