AUTUMN 2013

MODEL MIDSEMESTER SOLUTION

SUBHAM GHOSH 11CS10060

1)

a) Java language processors combine compilation and interpretation. A Java source program may first be compiled into an intermediate form called *bytecodes*. The bytecodes are then interpreted by a virtual machine. A benefit of this arrangement is that bytecodes compiled on one machine can be interpreted on another machine, perhaps across a network.

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 = '-', then both the situations are possible due to left associativity as both operators have same precedence.

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)

Automation	Initial	Per String
NFA	O(|r|)	O(|r| * |x|)
DFA typical case	O(|r|^3)	O(|x|)
DFA worst case	O(|r|^{2}2$^{|r|}$)	O(|x))

The above table summarizes the options when one is given a regular expression *r* and wants to produce a recognizer that will tell whether one or more strings *x* are in *L(r)*.

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

2)

i. Stmt -> if Exp then Stmt Stmt’ | a
Stmt’ -> else Stmt | E
Exp -> b

ii. First and Follow:-

	FIRST	FOLLOW
Stmt	if, a	$, else, E
Exp	b	then
Stmt’	else, E	$, else, E

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

(iii) The above predictive parsing table is conflicted in the rule (Stmt’, else) and hence is not LL(1).

(b) The following rule is not taken care of:

If A -> alpha | beta

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

3)

a) These grammars have the property that no production right can be epsilon or has the adjacent non terminals. The tokens are manipulated in such a way, that the grammar can be ignored and the nonterminals on the stack are used as placeholders for the attributes.

b)

Equivalent operator grammar:

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

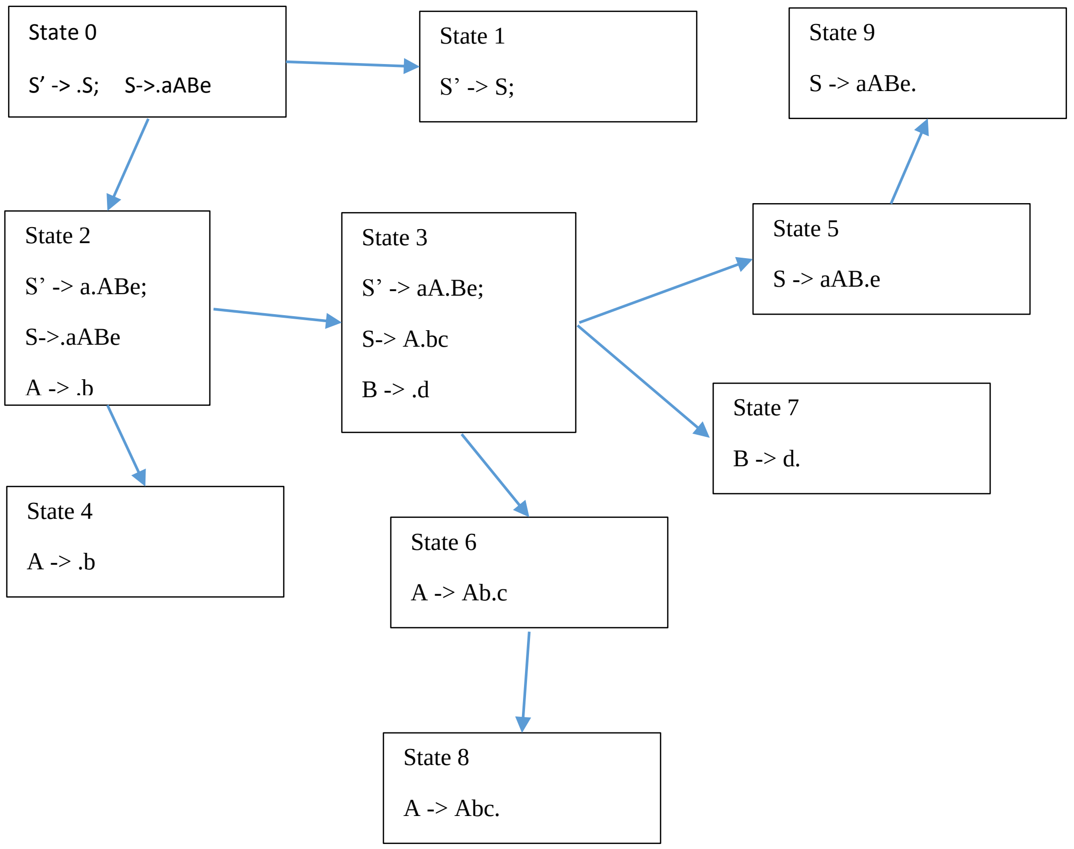
	Id	+	−	*	/	$
id	N/A	>	>	>	>	>
+	<	>	>	<	<	>
−	<	>	>	<	<	>
*	<	>	>	>	>	>
/	<	>	>	>	>	>
$	<	<	<	<	<	N/A

c)

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			