5.8 Consider the following grammar

declaration → type var-list

type → int | float

var-list → identifier, var-list | identifier

a. Rewrite it in a form more suitable for bottom-up parsing.

Solution:

declaration → type var-list

type → int | float

var-list → var-list, identifier | identifier

b. Construct the LR(0) DFA for the rewritten grammar.

Solution:

decl -> type vlist

type -> int

type ->float

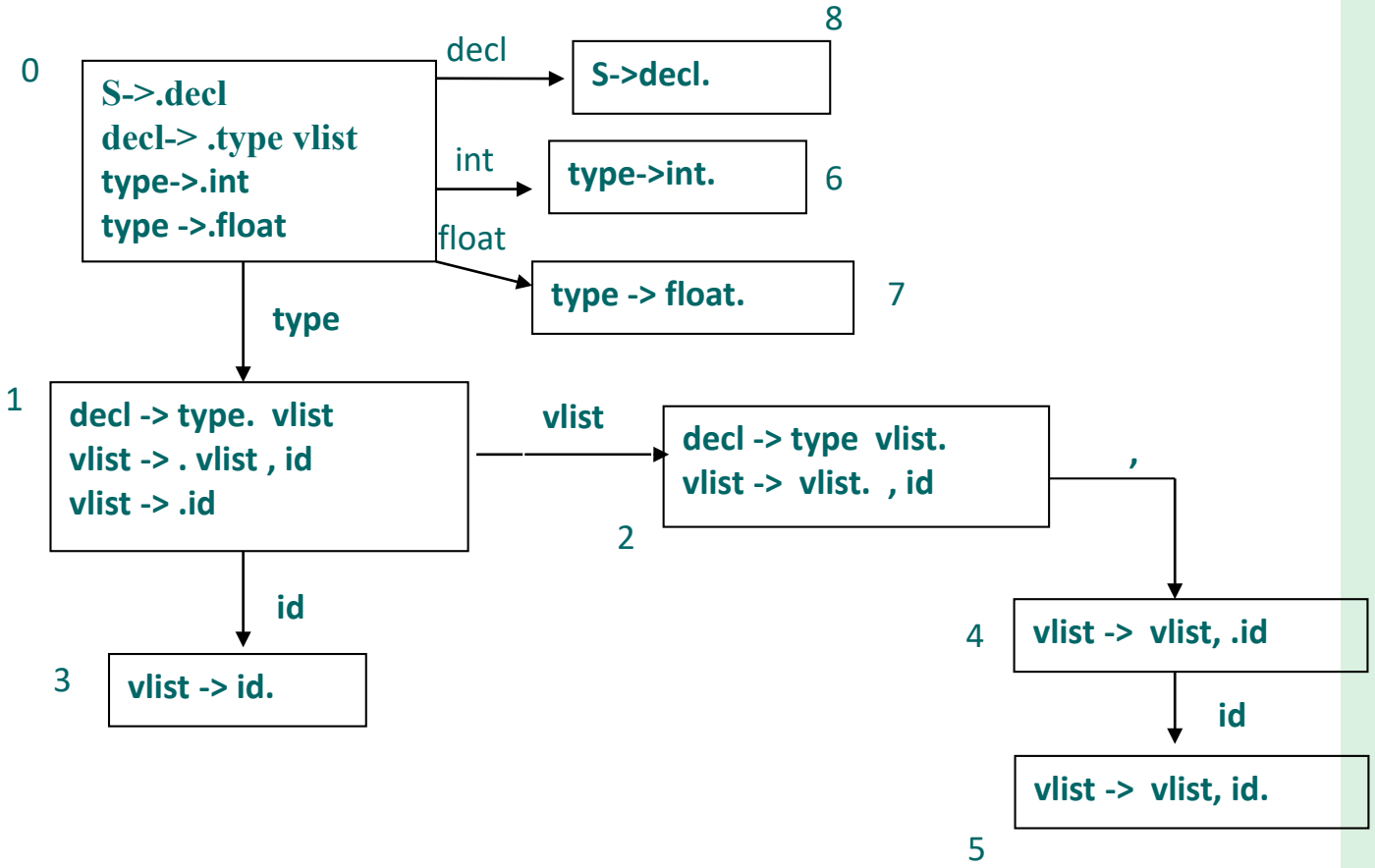
vlist -> vlist, id

vlist -> id

follow(decl)= { $ },

follow(type)= { id }

follow(vlist)= { $, , }



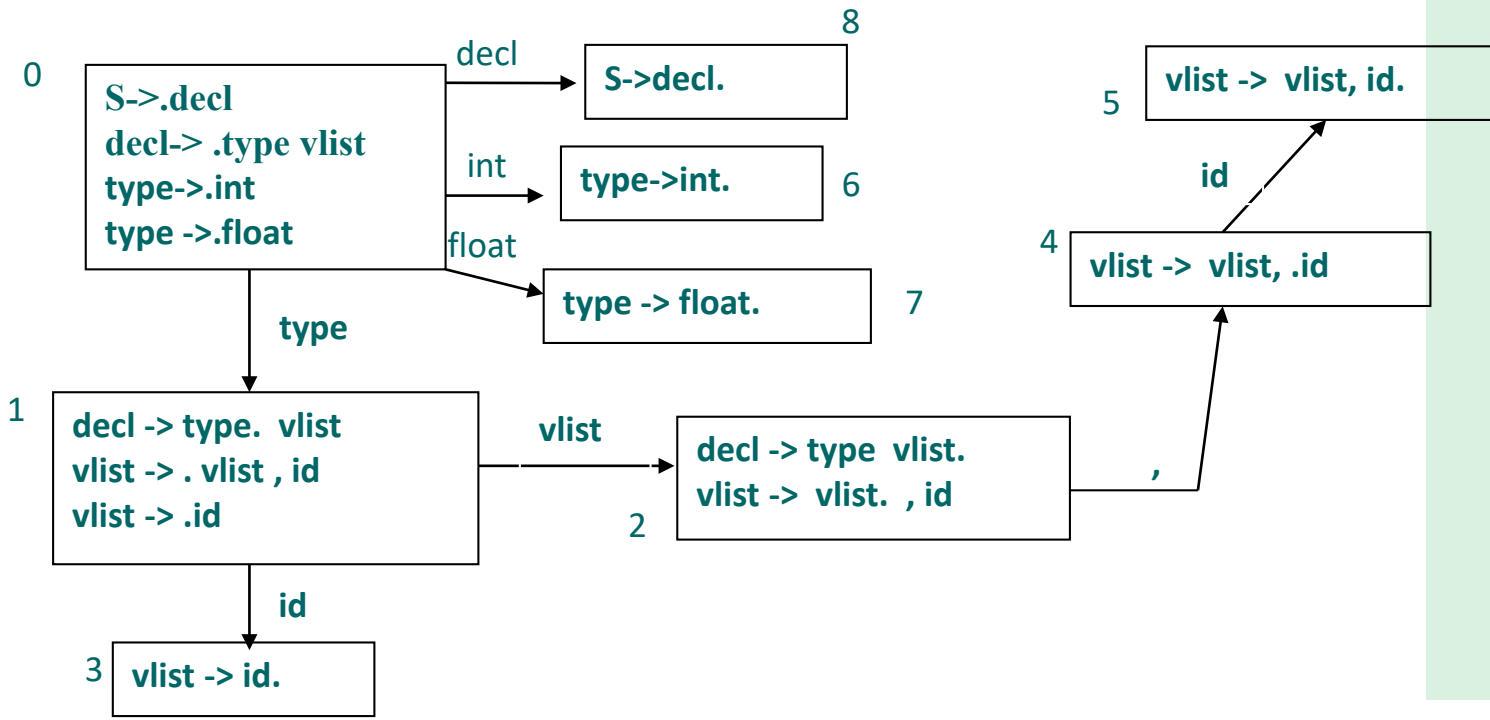
(c) Construct the SLR(1) parsing table for the rewritten grammar.

Solution:

state	input					goto		
	int	float	id	,	$	type	decl	vlist
0	s6	s7				g1	g8	
1			s3					g2
2				s4	r1			
3				r5	r5			
4			s5					
5				r4	r4			
6			r2					
7			r3					

decl -> type vlist
type -> int
type ->float
vlist -> vlist, id
vlist -> id

follow(decl)= { $ },
follow(type)= { id }
follow(vlist)= { $, , }



5.12 Show the following grammar is LR(1) but not LALR(1):

s→a A d | b B d | a B e | b A e

A→c

B→c

Solution:

- The augmented grammar is:

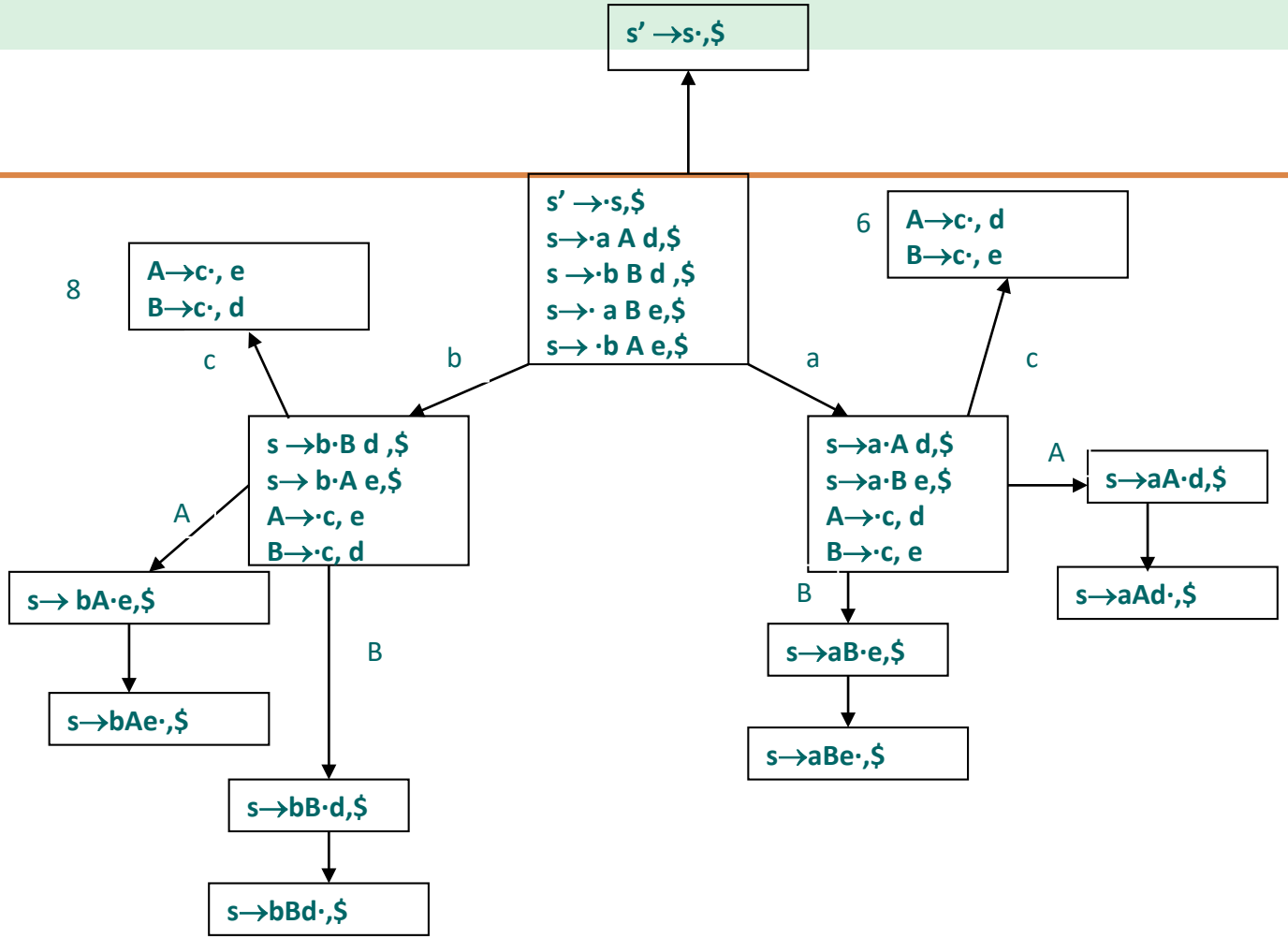
  s' →s

  s→a A d | b B d | a B e | b A e

  A→c

  B→c

The LR(1) DFA is shown as follows



To construct LALR(1) DFA, state 6 and state 8 are combined to a new state:

A→c·, d/e
B→c·, d/e

There is a reduce-reduce conflict in this state, which shows that the grammar is not LALR(1).