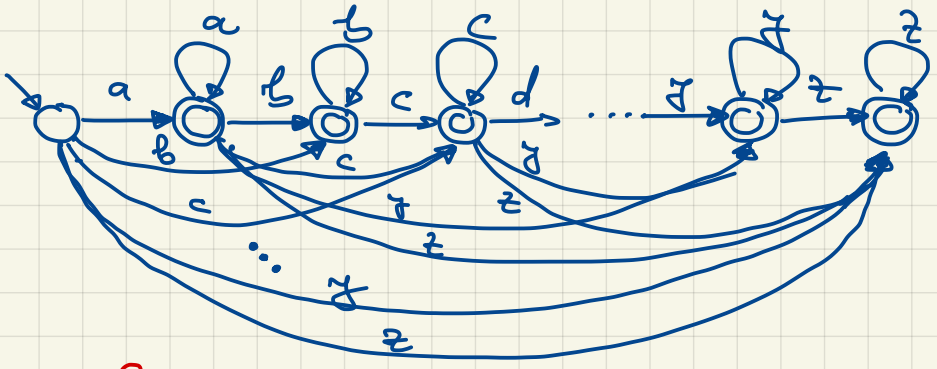
1 (10) (a) $a^+b^*\dots z^* + a^*b^+c^*\dots z^* + \dots + a^*b^*\dots j^*z^+$

give 8pt for the almost correct solution : $a^*b^*\dots c^*$

(b) (15)



-1 pt for each error

(2) (a) (10) -1pt for each error

X	FIRST(x) (3)	FOLLOW(x) (3)
P	a, b	$
S	a, b	$
A	a	b, c, $
B	b	c, $
C	c	c, $

p	(4) PREDICT(p)
0	a, b
1	a, b
2	a
3	b, c, $
4	b
5	b
6	c
7	c, $

(b) (5) Two conflicts: (2) $b \in PREDICT(B \to bB) \cap PREDICT(B \to bA)$

(3) $c \in PREDICT(C \to \varepsilon) \cap PREDICT(C \to Cc)$

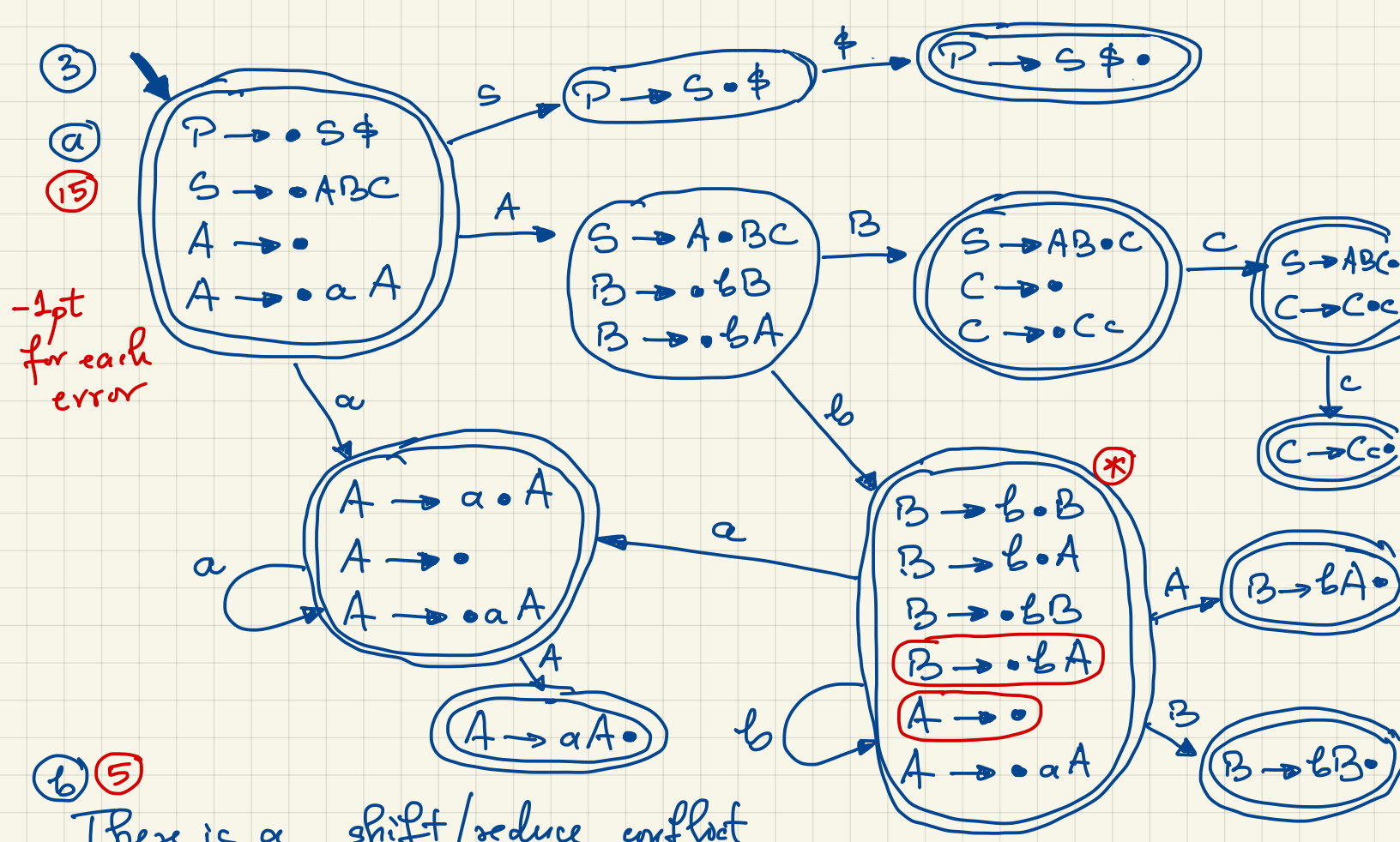
(c) (10) We modify G to correct both problems at (b):

$P \to S\$$
$S \to ABC$
$A \to aA$
$A \to \varepsilon$

$B \to bD$
$D \to A$
$D \to B$
$C \to cC$
$C \to \varepsilon$

(5) - common prefix removal:
$FIRST(A) \cap FIRST(B) = \emptyset$

(5) - by changing Cc into cC, C performs the same job but we no longer have $c \in FOLLOW(C)$.



b (5) There is a shift/reduce conflict in state (*):

$B \to \bullet\, bA$: shift on b to state (*)

$A \to \bullet$: reduce on $b \in \text{FOLLOW}(A)$ by production $A \to \varepsilon$

c (5) shift/shift conflicts do not exist by the very definition of the graph: when the dot '•' is followed by the same symbol in two LR-items of the same state, it will shift past the symbol and the two new LR-items will stay together in the new state.

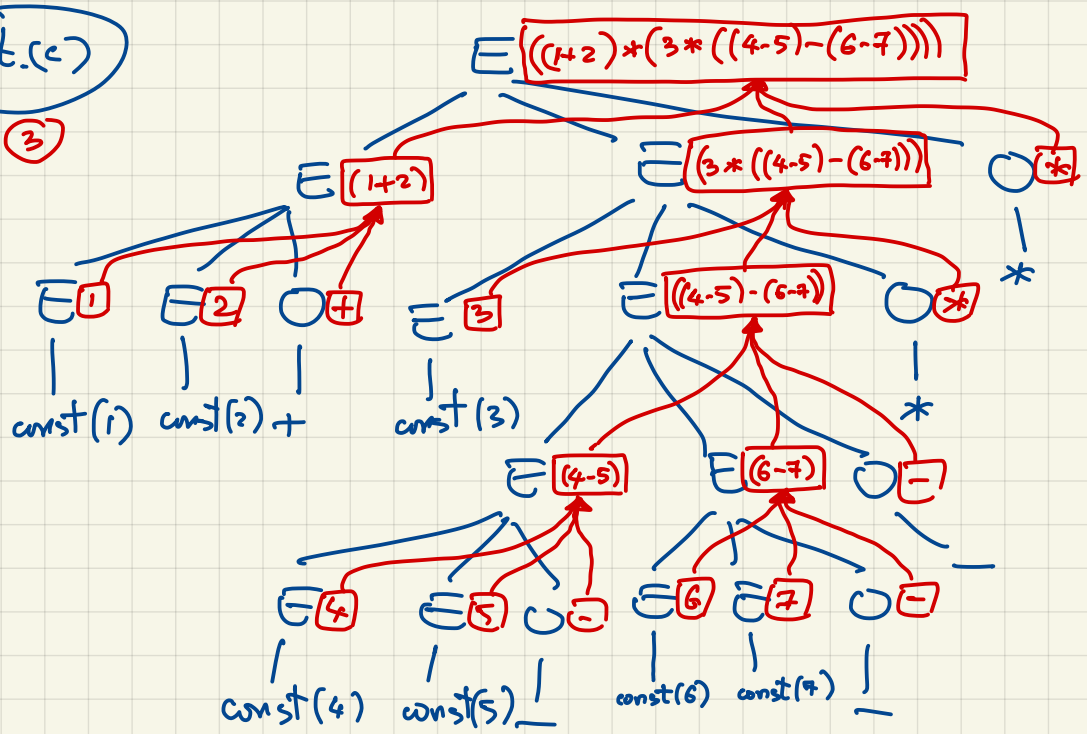
(4) (a) $E_1 \rightarrow E_2 E_3 O$ $\quad E_1.in = '(' + E_2.in + O.in + E_3.in + ')'$

(10) $E \rightarrow const$ $\quad E.in = const.string$

$O \rightarrow + | - | * | /$ $\quad O.in = '+' \mid '-' \mid '*' \mid '/'$

Alt. (c)

(3)



in stores the infix expression

(6) (10)

in = infix expr.
op = last operator used
prec(op) = precedence ; prec(*) = prec(/) > prec(+) = prec(-)

$E_1 \rightarrow E_2\ E_3\ O$

(...) ... (...)
LL LR RL RR

LL = LR = RL = RR = ε

if (E_2.op ≠ ε) and (prec(E_2.op) < prec(O.in)) then
    (LL, LR) = ('(', ')')

(5) — handling precedence

if (E_3.op ≠ ε) and
    ((prec(E_3.op) < prec(O.in)) or
    (prec(E_3.op) == prec(O.in)) and
    (O.in = '-' or O.in = '/')) then
    (RL, RR) = ('(', ')')

(5) handling associativity

E_1.in = LL + E_2.in + LR + O.in + RL + E_3.in + RR

E_1.op = O.in

E → const E.in = const.string
E.op = ε

O → + | - | * | / O.in = '+' | '-' | '*' | '/'

(c)

(5)

in – infix

up – last operator

prec(up) – precedence