A2 sol

Q1 (a) G_1: 1. $P \to S\$\$$
2. $S \to (S)S$
3. $S \to [S]S$
4. $S \to \varepsilon$

	FIRST	FOLLOW
P	(, [, $$	∅
S	(, [	$$,),]

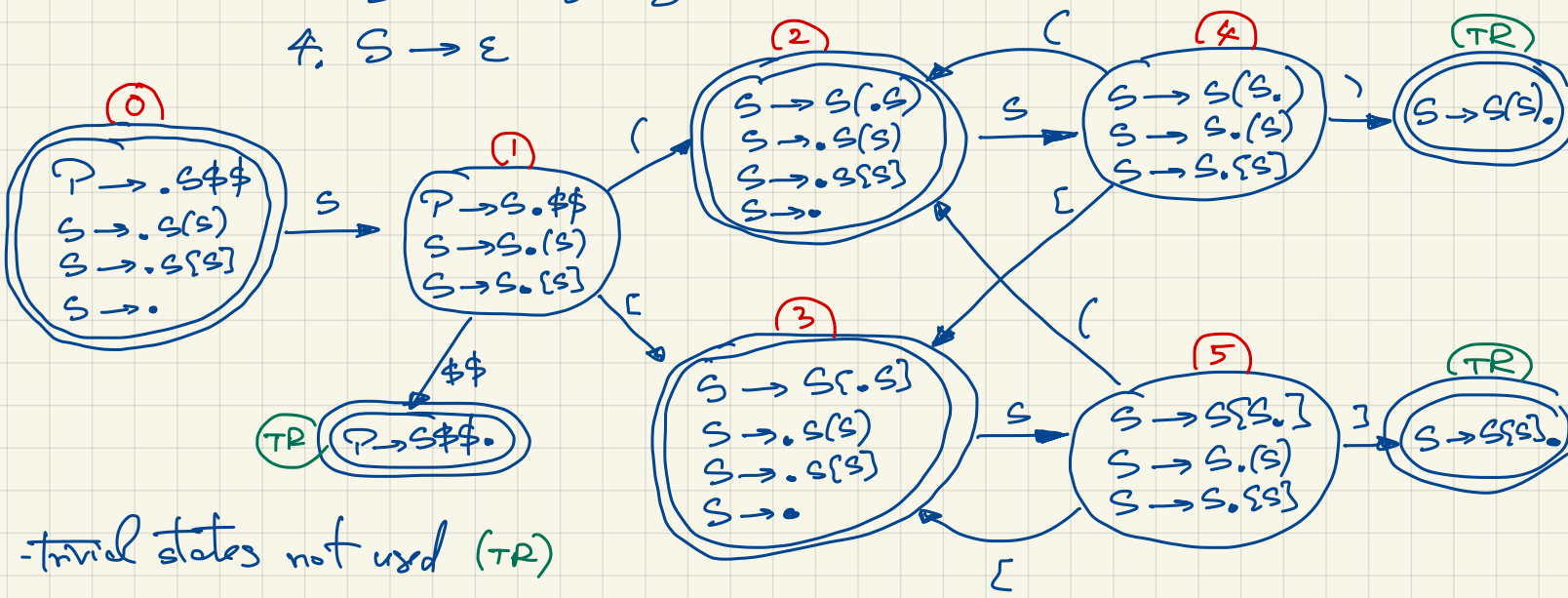
G_1 is LL(1) because each cell in the parse table has at most one entry.

Parse table:

nonterm. \ input	(	)	[	]	$$
P	1		1		1
S	2	4	3	4	4

(b) G_2: 1. $P \to S\$\$$
2. $S \to S(S)$
3. $S \to S[S]$
4. $S \to \varepsilon$

} → not LL(1)



- trivial states not used (TR)

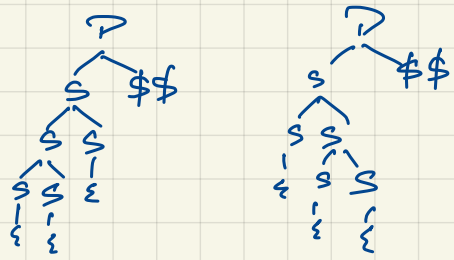
Parse table:

state \ input	S	(	)	[	]	$$
0	s1	r4	r4	r4	r4	r4
1		s2		s3		b1
2	s4	r4	r4	r4	r4	r4
3	s5	r4	r4	r4	r4	r4
4		s2	b2	s3		
5		s2		s3	b3	

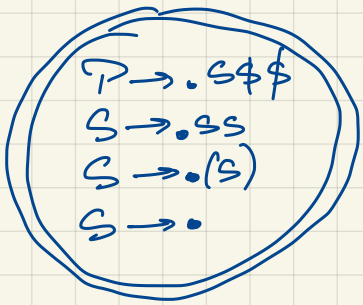
Thus is no conflict, so G_2 is SLR(1)

(c) G_3: $P \to S\$\$$
$S \to SS$
$S \to (S)$
$S \to \varepsilon$

G_3 is ambiguous : therefore not SLR(1)

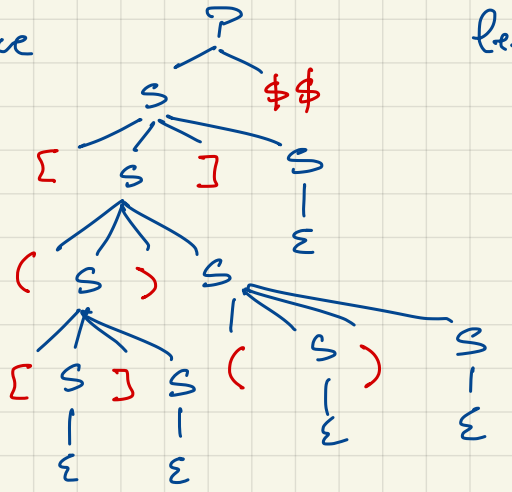


Alternative proof for not SLR(1):



- shift/reduce conflict on '('
  - shift: $S \to (.S)$
  - reduce: $S \to .$, '(' ∈ FOLLOW(S)

(d) G, parse tree



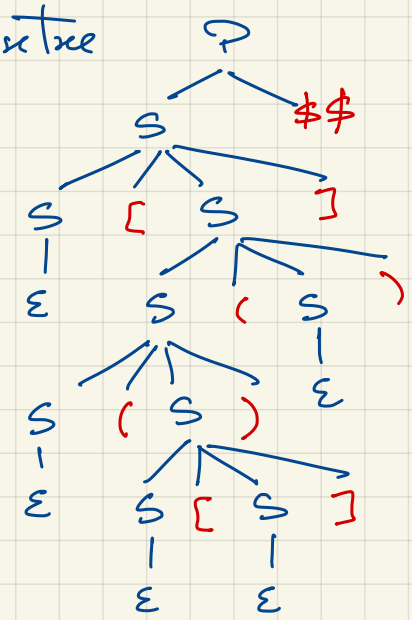
left derivation:

$P \Rightarrow S\$\$ \Rightarrow [S]S\$\$ \Rightarrow [(S)S]S\$\$$
$\Rightarrow [[(S)S]S]S\$\$ \Rightarrow [[]S]S\$\$$
$\Rightarrow [[]S]S\$\$ \Rightarrow [()]S]S\$\$$
$\Rightarrow [[]()]S\$\$ \Rightarrow [()]()]S\$\$$
$\Rightarrow [()]()]\$\$$

(e)

Parse stack	Input stream	Comment
P	[([])()]$$	
S$$	[([])()]$$	predict P → S$$
[S]S$$	[([])()]$$	S → [S]S
S]S$$	([])()]$$	match [
(S)S]S$$	([])()]$$	predict S → (S)S
S)S]S$$	[])()]$$	match (
[S]S)S]S$$	[])()]$$	predict S → [S]S
S]S)S]S$$	])()]$$	match [
]S)S]S$$	])()]$$	predict S → ε
S)S]S$$	)()]$$	match]
)S]S$$	)()]$$	predict S → ε
S]S$$	()]$$	match
(S)S]S$$	()]$$	predict S → (S)S
S)S]S$$	)]$$	match (
)S]S$$	)]$$	predict S → ε
S]S$$	]$$	match)
]S$$	]$$	predict S → ε
S$$	$$	match]
$$	$$	predict S → ε
		match $$

(f) G_2 parse tree



right derivation:

$P \Rightarrow S\$\$ \Rightarrow S[S]\$\$$

$\Rightarrow S[S(S)]\$\$ \Rightarrow S[S()]\$\$$

$\Rightarrow S[S(S)()]\$\$ \Rightarrow S[S(S[S])()]\$\$$

$\Rightarrow S[S(S[])()]\$\$ \Rightarrow S[S([])()]\$\$$

$\Rightarrow S[([])()]\$\$ \Rightarrow [([])()]\$\$$

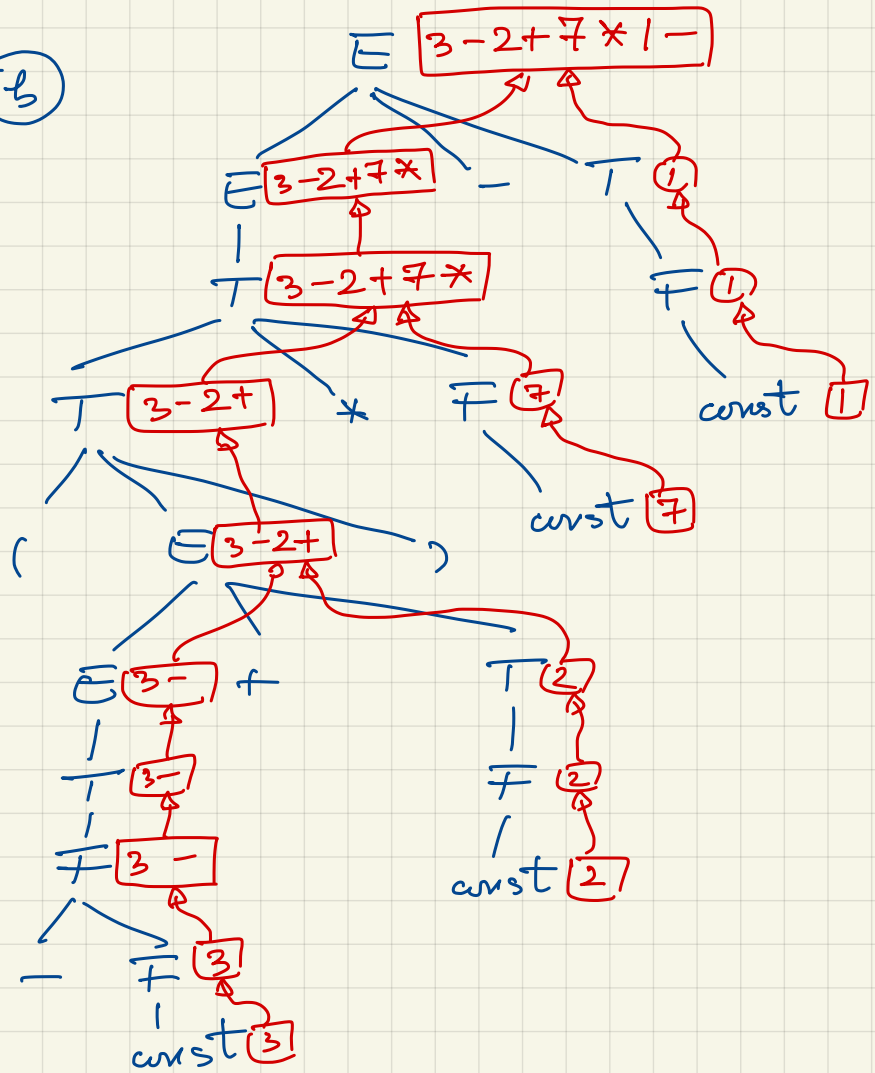
(g)

Parse stack	Input stream	Comment
0	[([])()]$$	reduce by S → ε
0	S[([])()]$$	shift S
0S1	[([])()]$$	shift [
0S1[3	([])()]$$	reduce by S → ε
0S1[3	S([])()]$$	shift S
0S1[3S5	([])()]$$	shift (
0S1[3S5(2	[])()]$$	reduce by S → ε
0S1[3S5(2	S[])()]$$	shift S
0S1[3S5(2S4	[])()]$$	shift [
0S1[3S5(2S4[3	])()]$$	reduce by S → ε
0S1[3S5(2S4[3	S])()]$$	shift S
0S1[3S5(2S4[3S5	])()]$$	shift and reduce by S → S[S]
0S1[3S5(2	S)()]$$	shift S
0S1[3S5(2S4	)()]$$	shift and reduce by S → S(S)
0S1[3	S()]$$	shift S
0S1[3S5	()]$$	shift (
0	)]$$	reduce by S → ε
0S1	S)]$$	shift S
0	)]$$	shift and reduce by S → S(S)
	S]$$	shift S
	]$$	shift and reduce by S → S[S]
	S$$	shift S
	$$	shift and reduce by P → S$$
	P	

Q2 (a) We use one synthesized attribute s, for the string associated with a node.

$E_1 \to E_2 + T$	▷ $E_1.s = \text{concat}(E_2.s, T.s, '+')$
$E_1 \to E_2 - T$	▷ $E_1.s = \text{concat}(E_2.s, T.s, '-')$
$E \to T$	▷ $E.s = T.s$
$T_1 \to T_2 * F$	▷ $T_1.s = \text{concat}(T_2.s, F.s, *)$
$T_1 \to T_2 / F$	▷ $T_1.s = \text{concat}(T_2.s, F.s, '/')$
$T \to F$	▷ $T.s = F.s$
$F_1 \to -F_2$	▷ $F_1.s = \text{concat}('-', F_2.s)$
$F \to (E)$	▷ $F.s = E.s$
$F \to \text{const}$	▷ $F.s = \text{str}(\text{const})$ (or just const)

(b)



(c) We use one synthesized attribute, s, for string and an inherited one, p, for partial string.

$E \to T\ TT$ ▷ $TT.p = T.s$ ▷ $E.s = TT.s$

$TT_1 \to +\ T\ TT_2$ ▷ $TT_2.p = concat(TT_1.p, T.s, '+')$ ▷ $TT_1.s = TT_2.s$

$TT_1 \to -\ T\ TT_2$ ▷ $TT_2.p = concat(TT_1.p, T.s, '-')$ ▷ $TT_1.s = TT_2.s$

$TT \to \varepsilon$ ▷ $TT.s = TT.p$

$T \to F\ FT$ ▷ $FT.p = F.s$ ▷ $T.s = FT.s$

$FT_1 \to *\ F\ FT_2$ ▷ $FT_2.p = concat(FT_1.p, F.s, '*')$ ▷ $FT_1.s = FT_2.s$

$FT_1 \to /\ F\ FT_2$ ▷ $FT_2.p = concat(FT_1.p, F.s, '/')$ ▷ $FT_1.s = FT_2.s$

$FT \to \varepsilon$ ▷ $FT.s = FT.p$

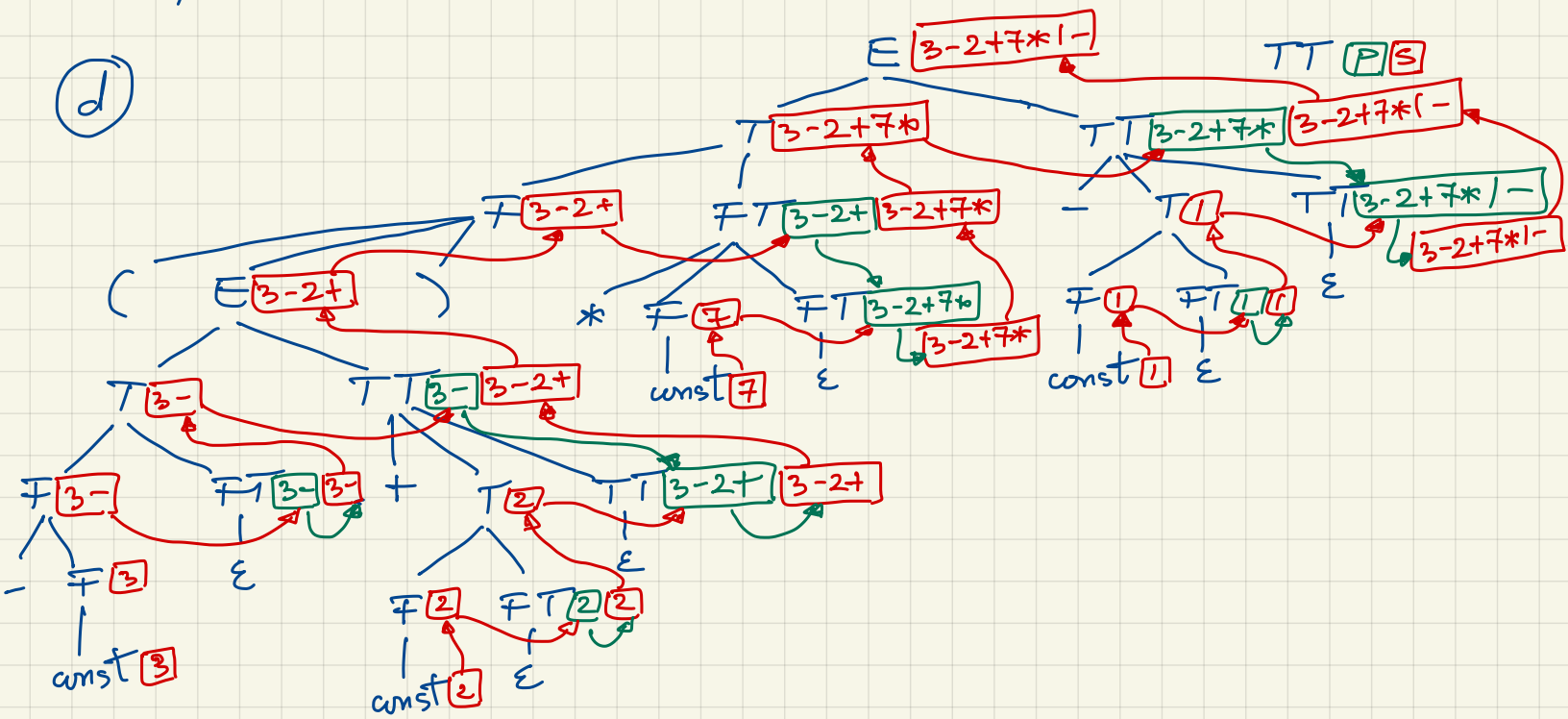
$F_1 \to -\ F_2$ ▷ $F_1.s = concat(F_2.s, '-')$

$F \to (\ E\ )$ ▷ $F.s = E.s$

$F \to const$ ▷ $F.s = str(const)$

$(-3+2)*7-1$

(d)



Q3 @ Besides the val attribute, we'll use pos for the position of the digit.

Float → Left . Right	▷ Float.val = Left.val + Right.val Right.pos = 0
Left → Digit Left_more	▷ Left.val = Digit.val + Left_mor.val Left.pos = Left_mor.pos + 1 Digit.pos = Left_mor.pos
Left_more → Left	▷ Left_mor.val = Left.val Left_mor.pos = Left.pos
Left_more → ε	▷ Left_mor.val = Left_mor.pos = 0
Right → Digit Right_more	▷ Right.val = Digit.val + Right_mor.val Right_mor.pos = Right.pos − 1 Digit.pos = Right_mor.pos
Right_more → Right	▷ Right_mor.val = Right.val Right.pos = Right_mor.pos
Right_more → ε	▷ Right_mor.val = 0
Digit → i	▷ Digit.val = i · $10^{Digit.pos}$

(b)