

CS 6353 Compiler Construction, Homework #2

1. Construct a type-0/1 grammar for the language  $\delta c \delta$ , where  $\delta$  can be any string of a's and b's. Use some examples to illustrate how your grammar would work.

$S \rightarrow ACME \mid BCNE$

$C \rightarrow ACM \mid BCN$

$MA \rightarrow AM$

$MB \rightarrow BM$

$NA \rightarrow AN$

$NB \rightarrow BN$

$ME \rightarrow AE$

$NE \rightarrow BE$

$A \rightarrow a$

$B \rightarrow b$

$C \rightarrow c$

$E \rightarrow \epsilon$

Generate  $\delta c \delta^{-1}$  with M equivalent to A and N equivalent to B  
An E at the end to help with the conversion of M to A and N to B

$S \Rightarrow ACME \Rightarrow ABCNME \Rightarrow ABBCNNME$

Convert the last symbol, M to A or N to B, and move it to the front

In parallel, other symbols can be converted and moved, but does not matter

$ABBCNNME \Rightarrow ABBCNNAE \Rightarrow ABBCNANE \Rightarrow ABBCANNE$

Continue with the conversion and move

$ABBCANNE \Rightarrow ABBCANBE \Rightarrow ABBCABNE \Rightarrow ABBCABBE$

The rest is to substitute nonterminals to terminals

2. Consider the following grammar. Note that id, +, [, ], and “,” are terminals.

$E \rightarrow E + T \mid T$   
 $T \rightarrow id \mid id[] \mid id[X]$   
 $X \rightarrow E, E \mid E$

(a) Eliminate left recursion in the grammar.

$E \rightarrow TE'$   
 $E' \rightarrow +TE' \mid \epsilon$   
 $T \rightarrow id \mid id[] \mid id[X]$   
 $X \rightarrow E, E \mid E$

(b) Perform left factoring for the grammar.

$E \rightarrow TE'$   
 $E' \rightarrow +TE' \mid \epsilon$   
 $T \rightarrow idT'$   
 $T' \rightarrow [T'' \mid \epsilon$   
 $T'' \rightarrow ] \mid X]$   
 $X \rightarrow EX'$   
 $X' \rightarrow ,E \mid \epsilon$

(c) Compute the First set for all symbols in the grammar.

$First(X') = \{ , , \epsilon \}$   
 $First(X) = First(E) = \{id\}$   
 $First(T'') = \{ ], First(X) \} = \{ ], id \}$   
 $First(T') = \{ [, \epsilon \}$   
 $First(T) = \{id\}$   
 $First(E') = \{ +, \epsilon \}$   
 $First(E) = First(T) = \{id\}$

(d) Compute the Follow set for all non-terminals in the grammar.

$Follow(E) = \{ \$ \} + First(X') - \epsilon + Follow(X') = \{ \$, , , \}$   
 $Follow(E') = Follow(E) = \{ \$, , , \}$   
 $Follow(T) = First(E') - \epsilon + Follow(E') = \{ \$, , , , + \}$   
 $Follow(T') = Follow(T) = First(E') - \epsilon + Follow(E') = \{ \$, , , , + \}$   
 $Follow(T'') = Follow(T') = Follow(T) = \{ \$, , , , + \}$   
 $Follow(X) = \{ \}$   
 $Follow(X') = Follow(X) = \{ \}$

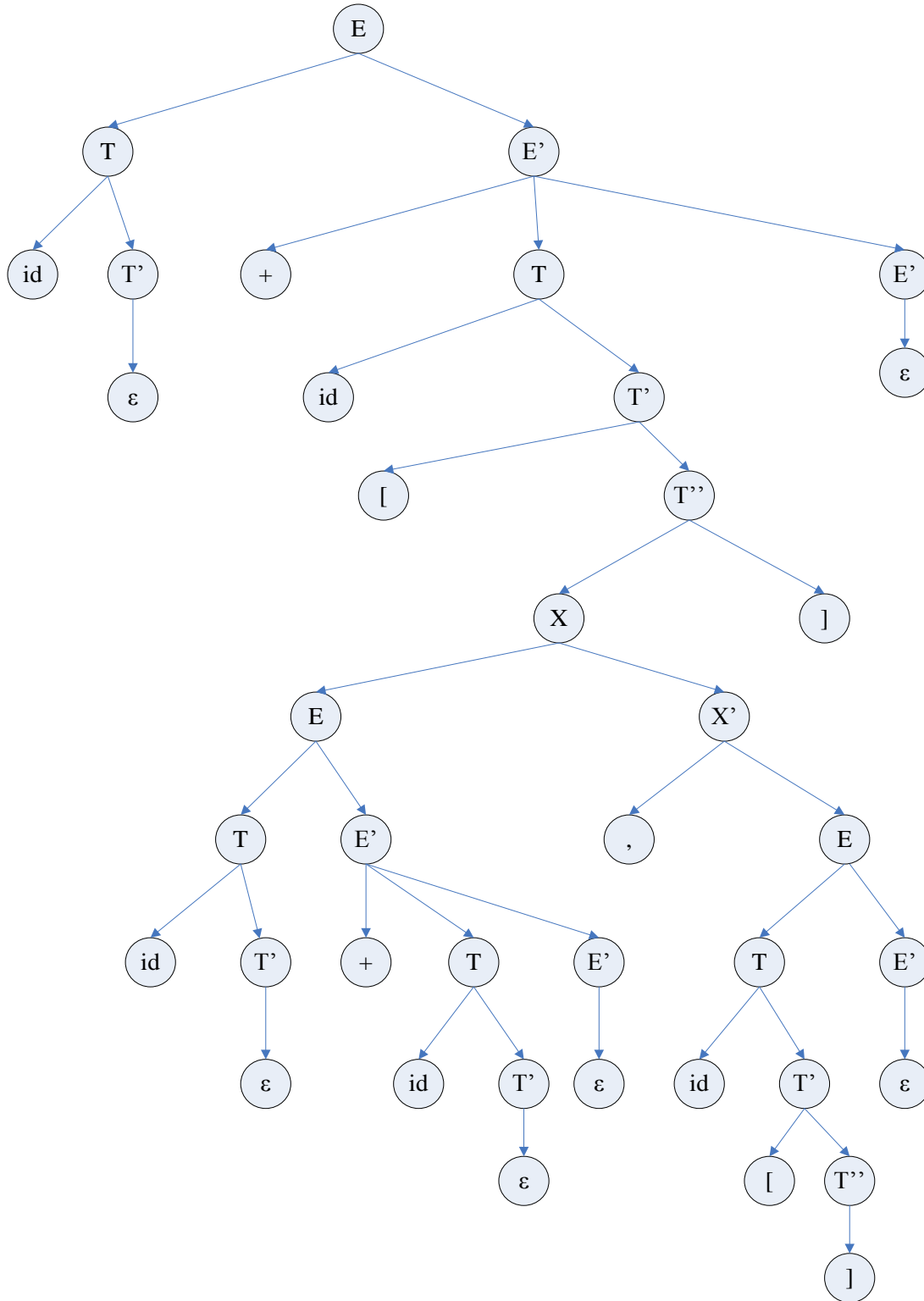
(e) Build an LL(1) parse table for the grammar.

	id	+	[	]	,	\$
E	$E \rightarrow TE'$					
E'		$E' \rightarrow +TE'$		$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow idT'$					
T'		$T' \rightarrow \epsilon$	$T' \rightarrow [T''$	$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
T''	$T'' \rightarrow X]$			$T'' \rightarrow ]$		
X	$X \rightarrow EX'$					
X'				$X' \rightarrow \epsilon$	$X' \rightarrow ,E$	

(f) Parse the string  $id + id[id+id, id[]]$ . Show the stack, the input, and the action taken.

Stack	Input	Action
E\$	id + id[id+id, id[]]\$	E->TE'
TE'\$	id + id[id+id, id[]]\$	T->idT'
T'E'\$	+ id[id+id, id[]]\$	T'-> $\epsilon$
E'\$	+ id[id+id, id[]]\$	E'->+TE'
TE'\$	id[id+id, id[]]\$	T->idT'
T'E'\$	[id+id, id[]]\$	T'->[T''
T''E'\$	id+id, id[]]\$	T''->X]
X]E'\$	id+id, id[]]\$	X->EX'
EX']E'\$	id+id, id[]]\$	E->TE'
TE'X']E'\$	id+id, id[]]\$	T->idT'
E'X']E'\$	+id, id[]]\$	E'->+TE'
TE'X']E'\$	id, id[]]\$	T->idT'
T'E'X']E'\$	, id[]]\$	T'-> $\epsilon$
E'X']E'\$	, id[]]\$	E'-> $\epsilon$
X']E'\$	, id[]]\$	X'->,E
E]E'\$	id[]]\$	E->TE'
TE']E'\$	id[]]\$	T->idT'
T'E']E'\$	[]]\$	T'->[T''
T''E']E'\$	]]\$	T''->]
E']E'\$	]\$	E'-> $\epsilon$
]E'\$	]\$	]->]
E'\$	\$	E'-> $\epsilon$
\$	\$	

(g) Build the parse tree while you are parsing. Show your parse tree



3. Consider the following grammar.

- $S \rightarrow As$
- $A \rightarrow BCA$
- $A \rightarrow BCa$
- $B \rightarrow b$
- $C \rightarrow c$

(a) Show that the grammar is not LL(1).

The rules  $A \rightarrow BCA$  and  $A \rightarrow BCa$ , has common left factors.

(b) Is the grammar LL(k)? If so, give the k value and show the parsing table.

$First_3(C) = \{c\epsilon\}$   $First_3(B) = \{b\epsilon\}$   $First_3(A) = \{bca, bcb\}$   $First(S) = \{bca, bcb\}$   
 $Follow_3(S) = \{\$\}$   $Follow_3(A) = \{s\$\}$   $Follow_3(B) = \{cbc, cas\}$   $Follow_3(C) = \{bca, bcb, as\$, \}$   
 Yes, it is LL(3).

In the table, we only list the useful columns, not all combinations of the input.

For S and A, only bca and bcb are useful

For B: we have  $b\epsilon$ , so look at the follow set of B and we have bcb and bca also.

For C: we have  $b\epsilon$ , with the follow(C), we have cbc and cas

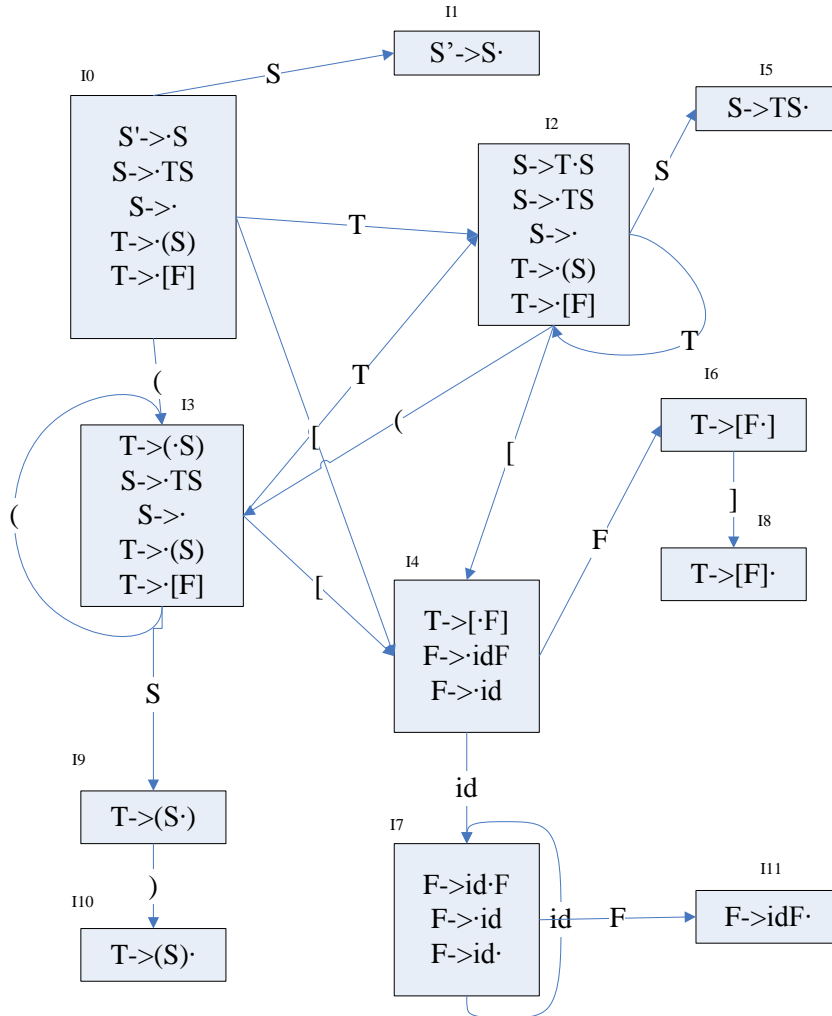
Thus, the table is:

	bca	bcb	cas	cbc
S	$S \rightarrow As$	$S \rightarrow As$		
A	$A \rightarrow BCa$	$A \rightarrow BCA$		
B	$B \rightarrow b$	$B \rightarrow b$		
C			$C \rightarrow c$	$C \rightarrow c$

4. Consider the following grammar. Note that id, (, and ) are terminals.

- $S \rightarrow TS \mid \epsilon$
- $T \rightarrow (S) \mid [F]$
- $F \rightarrow id F \mid id$

(a) Construct the LR(0) Automata for the grammar.



(b) Construct the SLR(1) parsing table for the grammar.

Follow(S) = { \$, }

Follow(T) = { (, [, ], \$ }

Follow(F) = { ] }

	id	(	[	]	)	\$	S	T	F
0		3	4		$S \rightarrow \epsilon$	$S \rightarrow \epsilon$	1	2	
1						Accept			
2		3	4		$S \rightarrow \epsilon$	$S \rightarrow \epsilon$	5	2	
3		3	4		$S \rightarrow \epsilon$	$S \rightarrow \epsilon$	9	2	
4	7								6
5					$S \rightarrow TS$	$S \rightarrow TS$			
6				8					
7	7			$F \rightarrow id$					11
8		$T \rightarrow [F]$	$T \rightarrow [F]$		$T \rightarrow [F]$	$T \rightarrow [F]$			
9					10				
10		$T \rightarrow (S)$	$T \rightarrow (S)$		$T \rightarrow (S)$	$T \rightarrow (S)$			
11				$F \rightarrow idF$					

(c) Parse the string  $([id\ id\ id]\ ()\ )\ [id\ id]$ . Show the stack, the input, and the actions taken.

Stack	Input	Action
0	$([id\ id\ id]\ ()\ )\ [id\ id]\ \$$	3
0(3	$[id\ id\ id]\ ()\ )\ [id\ id]\ \$$	4
0(3[4	$id\ id\ id]\ ()\ )\ [id\ id]\ \$$	7
0(3[4id7	$id\ id]\ ()\ )\ [id\ id]\ \$$	7
0(3[4id7id7	$id]\ ()\ )\ [id\ id]\ \$$	7
0(3[4id7id7id7	$]\ ()\ )\ [id\ id]\ \$$	F->id Goto(7, F) = 11
0(3[4id7id7F11	$]\ ()\ )\ [id\ id]\ \$$	F->idF Goto(7,F) = 11
0(3[4id7F11	$]\ ()\ )\ [id\ id]\ \$$	F->idF Goto(4,F) = 6
0(3[4F6	$]\ ()\ )\ [id\ id]\ \$$	8
0(3[4F6 ] 8	$()\ )\ [id\ id]\ \$$	T-> [F] Goto(3,T) = 2
0(3T2	$()\ )\ [id\ id]\ \$$	3
0(3T2(3	$)\ )\ [id\ id]\ \$$	S-> $\epsilon$ Goto (3,S) = 9
0(3T2(3S9	$)\ )\ [id\ id]\ \$$	10
0(3T2(3S9)10	$)\ [id\ id]\ \$$	T-> (S) Goto(2,T) = 2
0(3T2T2	$)\ [id\ id]\ \$$	S-> $\epsilon$ Goto (2,S) = 5
0(3T2T2S5	$)\ [id\ id]\ \$$	S->TS Goto(2,S) = 5
0(3T2S5	$)\ [id\ id]\ \$$	S->TS Goto(3,S) = 9
0(3S9	$)\ [id\ id]\ \$$	10
0(3S9)10	$[id\ id]\ \$$	T->(S) Goto(0,T) = 2
0T2	$[id\ id]\ \$$	4
0T2[4	$id\ id]\ \$$	7
0T2[4id7	$id]\ \$$	7
0T2[4id7id7	$]\ \$$	F->id Goto(7,F) =11
0T2[4id7F11	$]\ \$$	F->idF Goto(4,F) = 6
0T2[4F6	$]\ \$$	8
0T2[4F6]8	$\$$	T->[F] Goto(2,T) = 2
0T2T2	$\$$	S-> $\epsilon$ Goto(2,S) = 5
0T2T2S5	$\$$	S->TS Goto(2,S) = 5
0T2S5	$\$$	S->TS Goto(0,S) = 1
0S1	$\$$	Accept

(d) This grammar is also LL, but not LL(1). Sketch a scheme that can process the grammar using LL. Do not use LL(2). Do not rewrite the grammar. Do it in LL(1) but with some add on rules.

	(	)	[	]	id	\$
S	$S \rightarrow TS$	$S \rightarrow \epsilon$			$S \rightarrow TS$	$S \rightarrow \epsilon$
T	$T \rightarrow (S)$		$T \rightarrow [F]$			
F					$F \rightarrow id F, id$ $F \rightarrow id, ]$	

The reason for not to use LL(2) is that LL(2) will require more space for the parsing table. What we can do is to lookahead an additional token only for the entry with conflict, which is [F,id]. Which of the two rules to be used will be determined by one additional lookahead token.

(e) Discuss the space requirement for SLR(1) and LL(1), including the parsing table size and the maximum parsing stack size for any input string.

SLR(1) parsing table has 6+3 columns and 11 rows while the LL(1) parsing table only has 6 columns and 3 rows. For some input string, LL stack will have more 's and ]'s, but SLR will have more id's in the stack. So the stack space requirements of LL and SLR depend on the input string patterns.

5. Consider the following grammar. Note that SS and PP are each a single non-terminal.

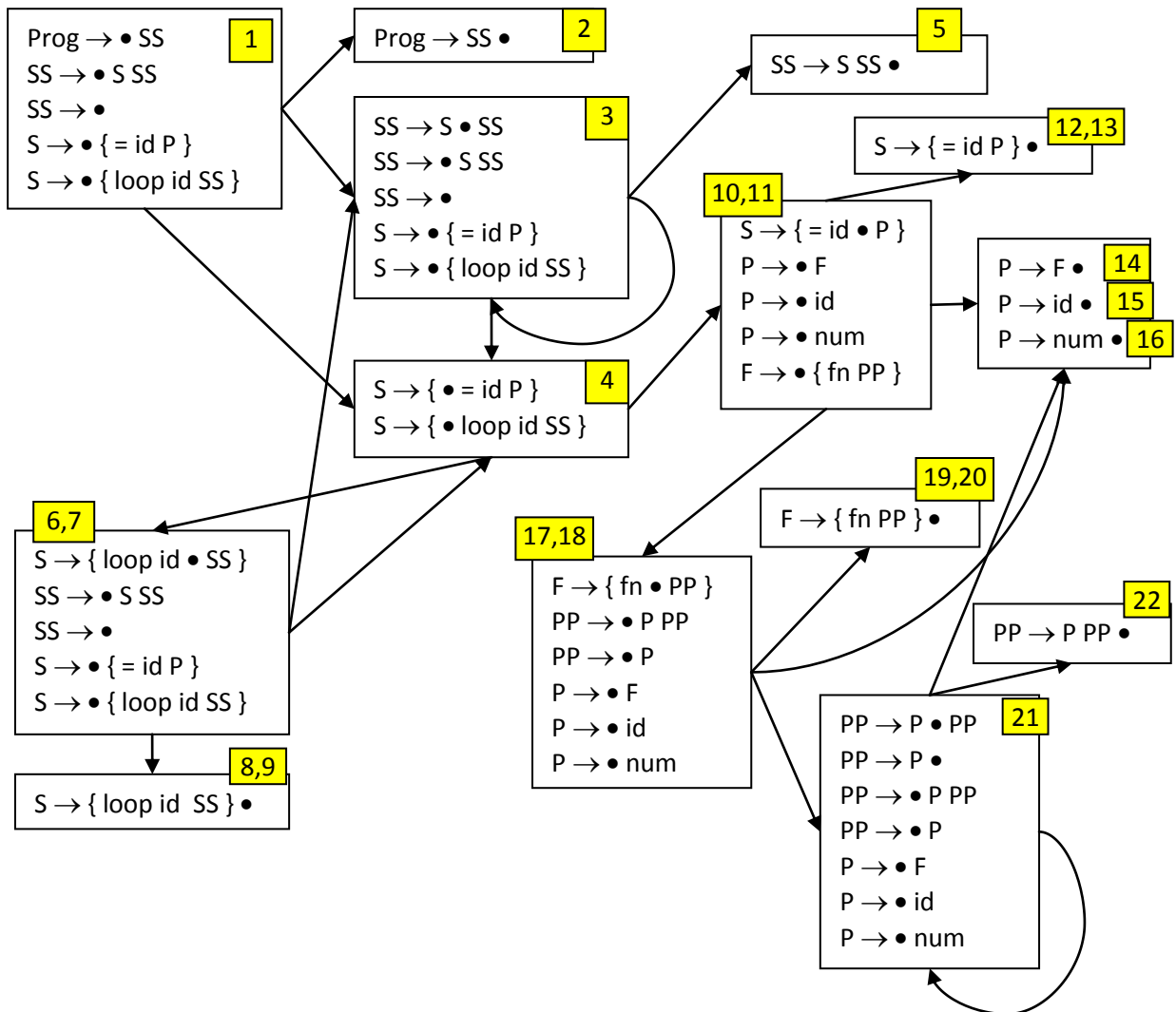
$Prog \rightarrow SS$   
 $SS \rightarrow S SS \mid \epsilon$   
 $S \rightarrow \{ = id P \}$   
 $S \rightarrow \{ loop id SS \}$   
 $P \rightarrow F \mid id \mid num$   
 $F \rightarrow \{ fn PP \}$   
 $PP \rightarrow P PP \mid P$

$First(SS) = \{ \epsilon \}$   
 $First(S) = \{ \}$   
 $First(PP) = \{ id, num \}$   
 $First(P) = \{ id, num \}$   
 $First(F) = \{ \}$

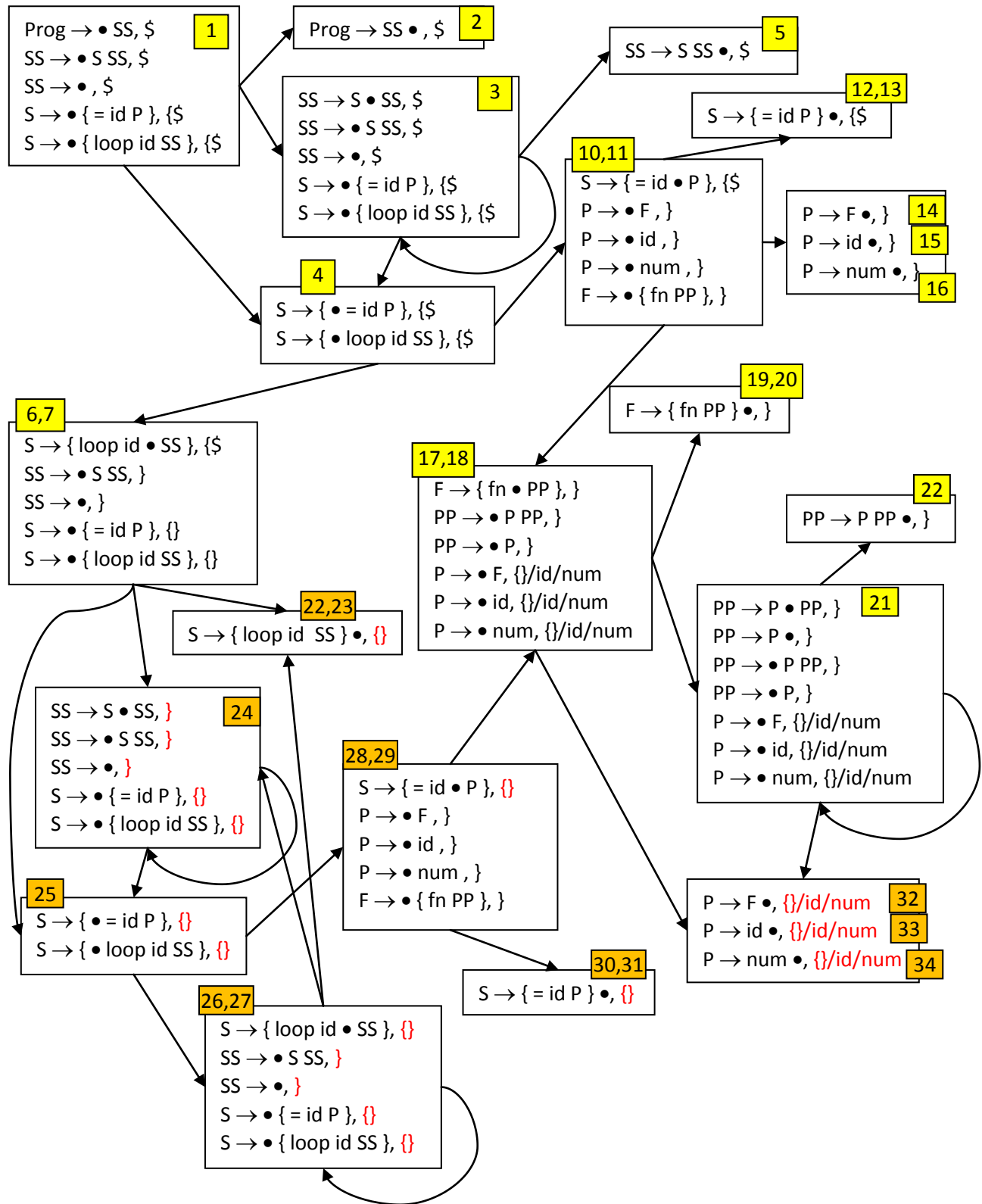
$Follow(SS) = \{ \$, \}$   
 $Follow(S) = \{ \$, \$, \}$   
 $Follow(PP) = \{ \}$   
 $Follow(P) = \{ id, num, \}$   
 $Follow(F) = \{ id, num, \}$

(a) Show that the grammar is not SLR(1).

There is no conflict. The grammar is SLR(1).



(b) Construct the LR(1) Automata for the grammar.



6. Consider the following grammar.

$S \rightarrow A-B)$

$A \rightarrow B \mid a)$

$B \rightarrow aB \mid a$

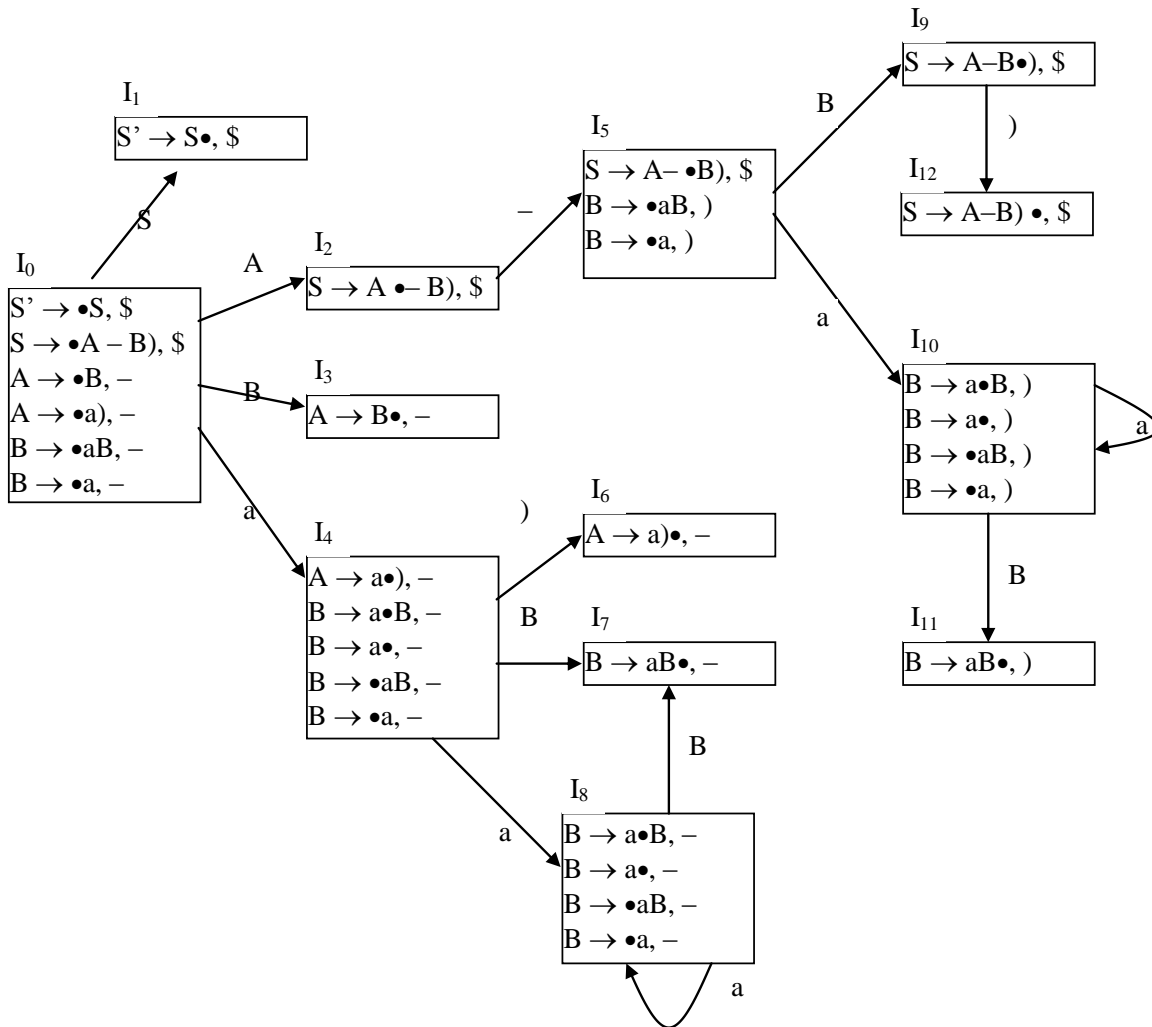
(a) Construct the LR(1) Automata for the grammar.

$S' \rightarrow S$

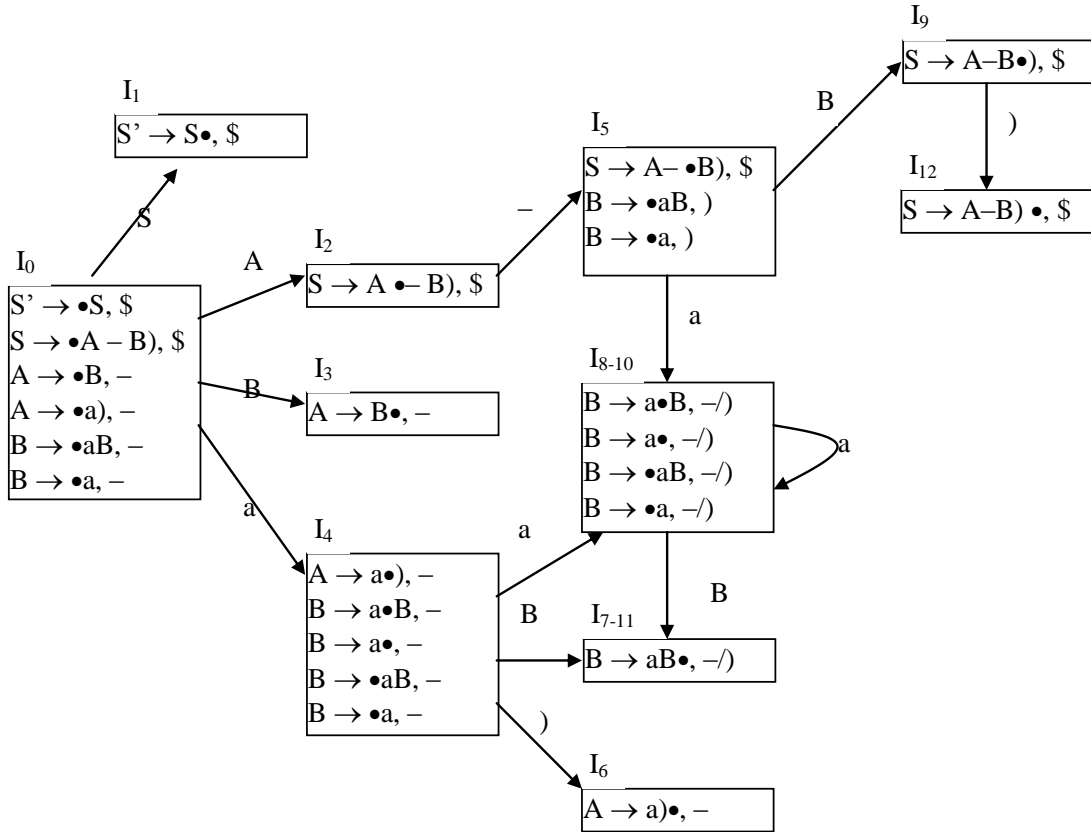
$S \rightarrow A-B)$

$A \rightarrow B \mid a)$

$B \rightarrow aB \mid a$



(b) Merge the LR(1) states and convert the LR(1) automata to LALR(1).



7. Give example grammars for the following specifications.

(a) A grammar that is not SLR(5) but is SLR(6).

$S \rightarrow A b^5 c \mid B b^5 d$

$A \rightarrow a$

$B \rightarrow a$

(b) A grammar that is LR(1), not LALR(1), and is LALR(2).

$S \rightarrow aAdx \mid bBdy \mid bAeu \mid aBev$

$A \rightarrow c$

$B \rightarrow c$

8. Assume that  $G$  is a context free grammar and  $G$  is also LL( $n$ ) and LR( $m$ ). Is it possible for  $n < m$ ? If so, show an example grammar that meets this criterion. If not, explain clearly why not.

If  $G$  is not LR( $m-1$ ), then there must exist a state where two LR( $m-1$ ) items having a conflict. There are two possible cases for the two LR( $m-1$ ) items.

Case 1: We have  $A \rightarrow \alpha \bullet \alpha'$  and  $B \rightarrow \beta \bullet$  in the same state and the two items are conflicting. We must have  $\alpha = \beta$ ; otherwise,  $A \rightarrow \alpha \bullet \alpha'$  and  $B \rightarrow \beta \bullet$  will not reach the same state. We can rewrite the items to  $A \rightarrow \alpha \bullet \alpha'$  and  $B \rightarrow \alpha \bullet$ . Also, we must have  $A \rightarrow \bullet \alpha \alpha'$  and  $B \rightarrow \bullet \alpha$  in the same state  $s$  (otherwise,  $A \rightarrow \alpha \bullet \alpha'$  and  $B \rightarrow \alpha \bullet$  will not reach the same state). If  $\alpha$  has  $A$  or  $B$  as its first symbol, then  $G$  is left recursive and is not LL. So,  $A$  and  $B$  are two different non-terminals. To have  $A \rightarrow \bullet \alpha \alpha'$  and  $B \rightarrow \bullet \alpha$  in  $s$ , we should have either of the following sets of additional rule(s): (a)  $A \rightarrow \bullet B \gamma$  or (b)  $B \rightarrow \bullet A \delta$  or (c)  $X \rightarrow \bullet B \gamma$  and  $X \rightarrow \bullet A \delta$ . In case (a) there will be overlapping first sets  $\text{First}_{m-1}$  for nonterminal  $A$  from  $A \rightarrow \bullet \alpha \alpha'$  and  $A \rightarrow \bullet B \gamma$ . In case (b) there will be overlapping first sets  $\text{First}_{m-1}$  for nonterminal  $B$  from  $B \rightarrow \bullet \alpha$  and  $B \rightarrow \bullet A \delta$ . Note that if  $\alpha$  is of length 0 or shorter than  $m-1$ , then the follow set should be used. The follow set of  $B$  should be the same as the first set of  $\alpha'$  for length  $m-1$  since  $G$  is LR( $m-1$ ). In case (c), there will be overlapping first sets  $\text{First}_{m-1}$  for nonterminal  $X$  from  $X \rightarrow \bullet B \gamma$  and  $X \rightarrow \bullet A \delta$ . Thus, in Case 1,  $n-1$  will be at least  $m-1$ .

Case 2: We have  $A \rightarrow \alpha \bullet$  and  $B \rightarrow \beta \bullet$  in the same state and the two items are conflicting. We must have  $\alpha = \beta$  and follow sets  $\text{Follow}_{m-1}$  of  $\alpha$  and  $\beta$  have overlap elements. Use the same proof as Case 1, we can conclude that the first sets  $\text{First}_{m-1}$  of one of the nonterminals  $A$ ,  $B$ , or  $X$ , will have overlapping elements from two production rules of that nonterminal. Thus,  $n \geq m$ .