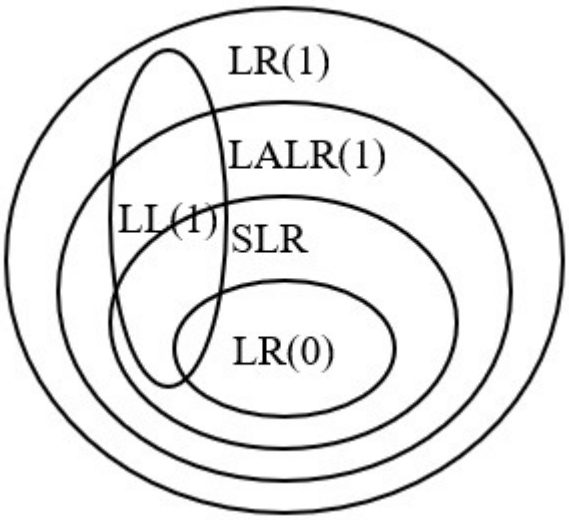
Quiz3: CS348 (full marks 25)

1) **Among LL(1), SLR, LR(1), LALR parsers, which one is the most powerful in terms of accepting more number of languages? If a grammar cannot be parsed by LALR parser, can it be parsed by LR(1)? Can it be parsed by LL(1)? (Marks: 1+1+1)**

   Ans:
   LR(1) parser is the most powerful.
   Yes it may be parsed by LR(1), but there is no guarantee (For example if the grammar is not even LR(1), i.e, in the diagram, it falls outside LR circle.)
   Yes it may be parsed by LL(1), but there is no guarantee.



2) **Why left recursion must be removed from LL(1) grammars? Remove left recursion from the following grammar. (Marks: 1+2)**

   S → A

   A → aB / Ad

   B → b

   C → g

   Ans:

   Left recursion needs to be removed from LL(1) because, in case of left recursive rules, when the non-terminal from the top of the stack is expanded with a let

recursive rule, the new top of the stack symbol again becomes the previous top of the stack symbol. As the input symbol is not advanced for non-terminal sysbol, we are again in the same situation with same top of stack symbol and same current input symbol. As a result, the algorithm goes into infinite loop.

S → A

A → aBA'

A' → dA' / ∈

B → b

C → g

3) **What is left factoring? Why it is essential for LL(1) parsing? Left-factor the following grammar. (Marks: 1+1+1)**

S → i E t S | i E t S e S | a

E → b

Ans:
A left-factored grammar is a class of grammar, where based on the current input symbol, a unique production of the given non-terminal symbol can be determined. The process of converting a non left-factored grammar to a left-factored grammar is called left-factoring.

S → iEtSS' / a

S' → eS / ∈

E → b

4) **Find FIRST() and FOLLOW() of all the nonterminal symbols of the following grammar. (Marks: 3+3)**

S → aBDh

B → cC

C → bC / ∈

D → EF

E → g / ∈

F → f / ∈

Ans:

First Functions-

- First(S) = { a }
- First(B) = { c }
- First(C) = { b , ∈ }
- First(D) = { First(E) – ∈ } ∪ First(F) = { g , f , ∈ }
- First(E) = { g , ∈ }
- First(F) = { f , ∈ }

Follow Functions-

- Follow(S) = { $ }
- Follow(B) = { First(D) – ∈ } ∪ First(h) = { g , f , h }
- Follow(C) = Follow(B) = { g , f , h }
- Follow(D) = First(h) = { h }
- Follow(E) = { First(F) – ∈ } ∪ Follow(D) = { f , h }
- Follow(F) = Follow(D) = { h }

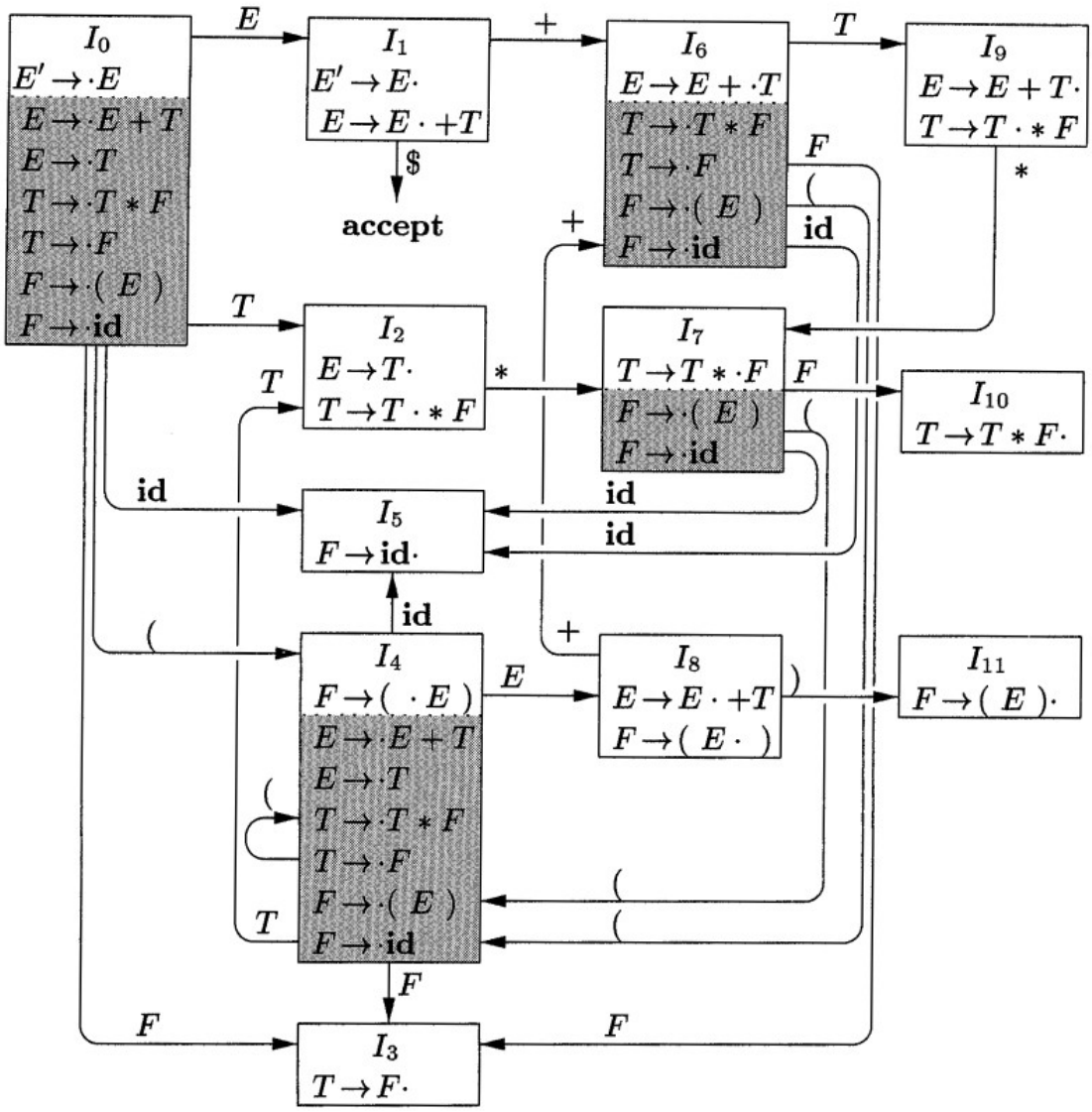
5) **Show the LR(0) item sets (automation) for the below grammar and construct the LR(0) parsing table for it. (Marks: 6+4)**

E → E + T | T

T → T * F | F

F → (E) | id

Ans: The parsing table shown here is of SLR type. For LR(0), all the columns of the reduce rows will have the reduce entry which may result in shift/reduce conflicts.



STATE	ACTION						GOTO		
	id	+	*	(	)	$	*E*	*T*	*F*
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			