Let’s write an SLR(1) parser for the following grammar for Prefix notation

E → n | A
A → (B)
B → +EE
B → *EE

(Don’t type the spaces!!)

What are the sets First(A), First(B) and First(E)?
*A must start with a ‘(‘ , B starts with either + or *, and E starts with a ‘(‘ or n*

What are the sets Follows(A),Follows(B) and Follows(E)?
A is followed by ‘$’ or ‘(‘ or ‘)’ or a number n
B is followed by a ‘)’
And since E reduces to a single number, which is followed by $, or what follows A, we have ‘$’ or ‘(‘ or ‘)’ or a number n.

For the first state q0, there are 4 possible transitions. It is possible we have a single number n$, in this case, the stack has the current state, 0, and we push the terminal ‘n’ onto the state with the next state state which is 4, the stack is now 4n0. We are at the end of input, ‘$’ so reading this we reduce by popping the state and the value from the stack, which brings us back to state 0. Having applied the reduction E→ n, we push the next state 3 and E onto the stack which is 3E0. Since the input pointer is at $ we accept the input.

Try the following input. (*(+nn)n) (don’t type any spaces!!). Let’s trace the steps. Reading a ‘(‘ we push the next state of 1 and the symbol ‘(‘ onto the stack which is now 1(0. The symbol ‘*’ causes us to shift to state 5 and both the next state and the symbol is pushed onto the stack which is now 5*1(0. In state 5 on reading a ‘(‘ we go to state 1, so we enter state 1 with the stack 1(5*1(0.

Reading the ‘+’ pushes us to state 6, so we enter this state with the stack 6+1(5*1(0. In state 6 we now read a n (a number) so we will go to state 4 with stack 4n6+1(5*1(0. In state 4 we read the next n which means we will reduce the n on the stack using rule E→n means we remove 4n from the stack (remember the top of the stack is the current state, so removing 4n moves us to state 6 with input E). In state 6 having the non-terminal E, causes us to move to state 9 with state 9E6+1(5*1(0. In state 8 we read the next n, which moves us to state 4 with stack. 4n9E6+1(5*1(0. Having the input pointer pointing at ‘)’ we reduce using E, which means we remove the 4n, and expose the 9 moving use to state 9 on an E so the stack will be 12E9E6+1(5*1(0. In state 12 we are pointing at a ‘)’ which means we reduce by using rule 4, so we will be going to state 1 (B→+EE, so 12E9E6+ are removed). Having entered state 1 using rule B, the next state with be 7, with the stack 7B1(5*1(0. In state 7, we need to read a closing ‘)’ for the add to be well formed. This drives us to state 10, with stack 10)7B1(5*1(0.

In state 10 with the input pointing to an n we reduce using rule 3, A → (B)… so we pop until we remove the ‘)’ bringing us to state 5 using A, so we will be entering state 2 with the stack 2A5*1(0. State 2 we read an n, so we reduce using rule 2, E→A, so we enter 5 with E, so we will go to state 8 with the stack being 8E5*1(0. In state 10, we shift, (input points to n) so we have 4n8E5*1(0 as the stack. In state 4 we reduce, since the input points to ‘)’, the rule being 1, so we enter state 8 with the rule being applied E, so we will enter state 11 with the stack 11E8E5*1(0.   ** Notice!! We are almost well formed, with an E E * being on the stack**.

In state 11 we the input points to a ‘)’ so we reduce using rule 5, entering state 1 via a B, so we will be going to state 7 with stack 7B1(0 and with the input pointing to a ‘)’, so we will shift to state 10 with the stack 10)7B1(0.

In state 10 we will reduce using rule 3 so we will enter state 0 using A→ (B) which takes us to state 2, with the input pointing to a $... with the stack 2A0. In state 2 we will reduce using rule 2, E→A which takes us to state 0 via an E, which returns us to state 3 with stack 3E0, and the input pointing to a $ so we accept it.