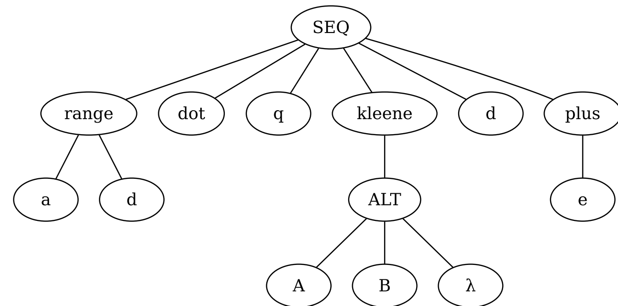
For this particular LGA, you will all work on the same problem. You will share your solutions (and most importantly the *method* of your solutions) during the next lecture period.

RE	→	*ALT* $
ALT	→	*SEQ ALTLIST*
ALTLIST	→	*pipe SEQ ALTLIST*
	|	λ
SEQ	→	*ATOM SEQLIST*
	|	λ
SEQLIST	→	*ATOM SEQLIST*
	|	λ
ATOM	→	*NUCLEUS ATOMMOD*
ATOMMOD	→	*kleene*
	|	*plus*
	|	λ
NUCLEUS	→	*open ALT close*
	|	*char CHARRNG*
	|	*dot*
CHARRNG	→	*dash char*
	|	λ

The regular expression `a-d.q(A|B|)*de+` from the LL(1) grammar below would yield the "RE Tree" (or **abstract syntax tree**, AST) if parsed with SDT processing procedures (`lga-re-sdt.pdf`).



The thought-exercise for this assignment is determining the algorithms for "handling" each particular node in the **simplified** RE parse tree — the final result will be the NFA for the regular expression. There are **eight** node types: `SEQ`, `ALT`, `range`, the `kleene` operator `*`, and leaf nodes for a character, any character (`dot`) and λ. Notice that the λ in the tree represents an empty character sequence, **it is not the same as the λs in the grammar** (but it did "evolve" from one of them).

As in `show_re-nfagen.pdf` , we will organize the NFA in conventional tabular (2d array) form (T). States are rows, characters are columns, and transitions are represented by table cells containing the destination state number. We will represent λ transitions with a square matrix L with rows representing source (originating) state and columns the destination state. There will be no entries along the main diagonal of L.

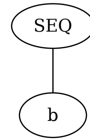
Envision an algorithm using in-order traversal of the **RE Tree**, each node (including leaves) are provided two parameters: *src* and *dest*. These represent the node's initial and final states.

The whole algorithm begins with two states (starting state 0 and accepting state 1), and each node may add states dynamically as the tree is traversed.

Your task: design your algorithms for the **eight** node types (in reasonable pseudo code) such that your group members can test your logic when you meet next.

Fair warning: the example logic in `show_re-nfagen.pdf` may not work with your logic for the balance of procedures you need. It should be clear that the logic of the algorithms you develop will have to work as a cohesive whole, with certain assumptions and invariants consistent throughout.

Example: Consider the trivial regular expression `b` in a language with characters `a`–`e`. This regular expression would result in the AST below. The root `SEQ` node logic from `show_re-nfagen.pdf` would call the `leafChar` **visitor pattern procedure** with the root node's *src* and a new $childdest = 2$ destination state.



The `leafChar` logic is close to a one liner:

```
leafChar( src, dest ) (
  v ← my leaf character value
  T[src][v] ← dest
)
```

And the tables T and L would end the algorithm as:

T	a	b	c	d	e
0		2			
+1					
2					

L	0	+1	2
0			
+1			
2			

(Where + means an accepting state.)