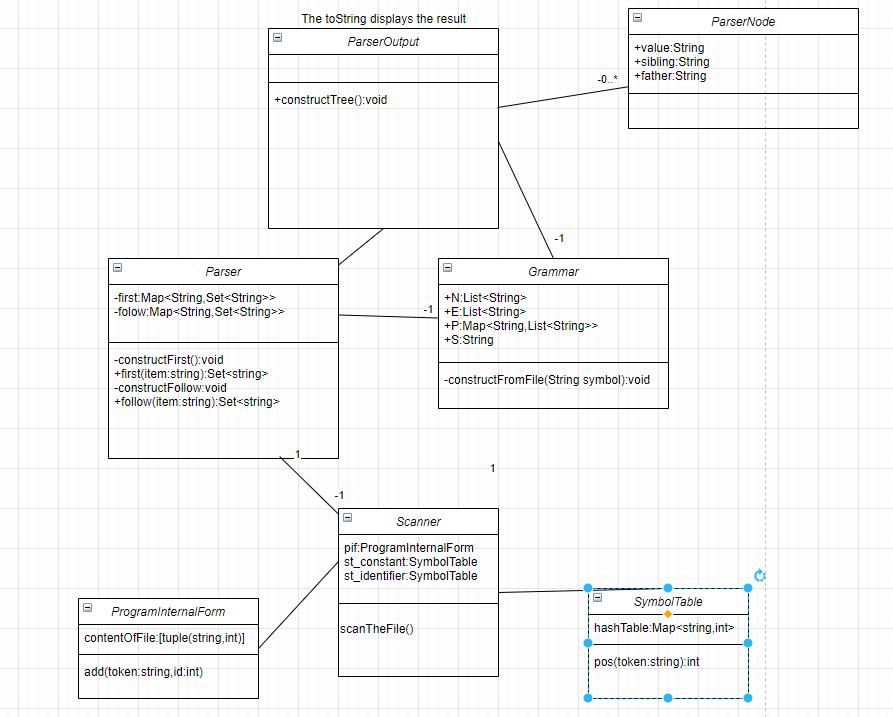
Ravasz Tamás, Andrei-Daniel Sălăgean

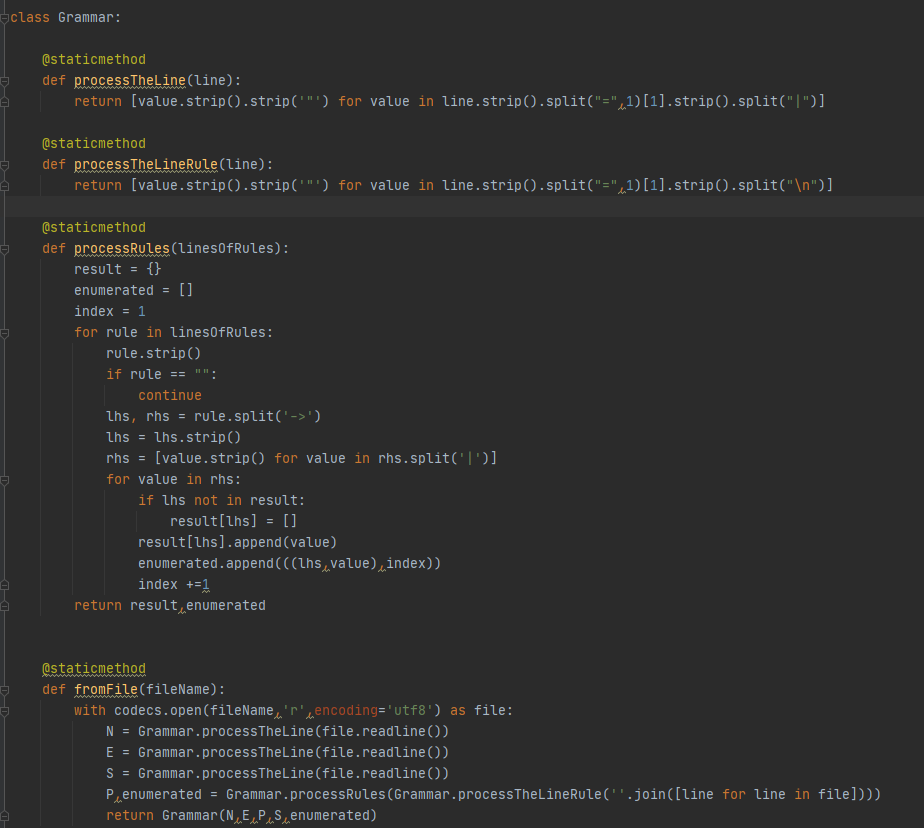
Parser documentation

<https://github.com/RavaszTamas/FormalLanguagesLaboratories/tree/main/lab5-7>

Class diagram



Class diagram for the application



The main representation of a grammar used by our application.

N = "S" | "A" | "B" | "C" | "D"  
E = "+" | "ɛ" | "\*" | ")" | "(" | "a"  
S = "S"  
P =  
S -> B A  
A -> + B A | ɛ  
B -> D C  
C -> \* D C | ɛ  
D -> ( S ) | a

An example like this is used here where the nonterminals and terminals are in sperarate rows and the starting symbol is defined in a separate line then each production in a separate line.



FIRST algorithm. It follows what was described in the professor’s description only that not all sets are checked but the a boolean value is observed to check wheter any changes occur.

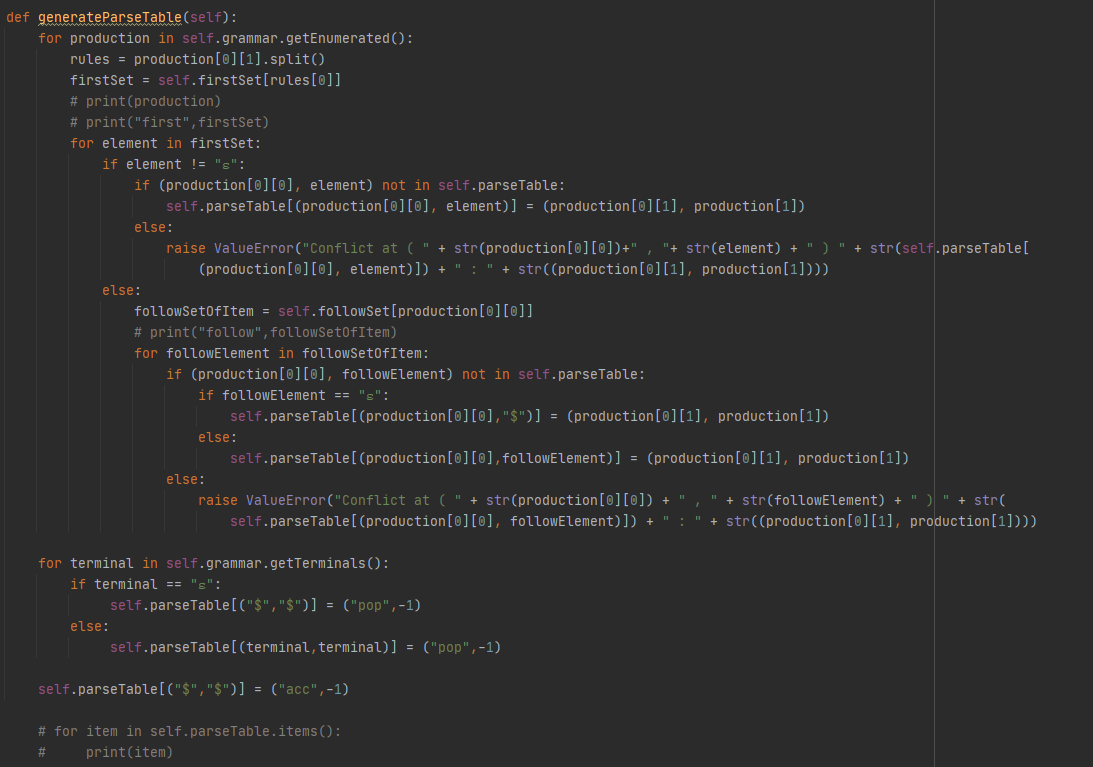
At the initalization the all terminals get the first set as themselves then all non-terminals will get the the first terminal that appears as the first item in each production where the non-terminal is at the left hand side. After that for each non terminal if the for all items on the right hand side we have a non-empty first set then we will generate in the concat result is the result of the concatenation of length one.



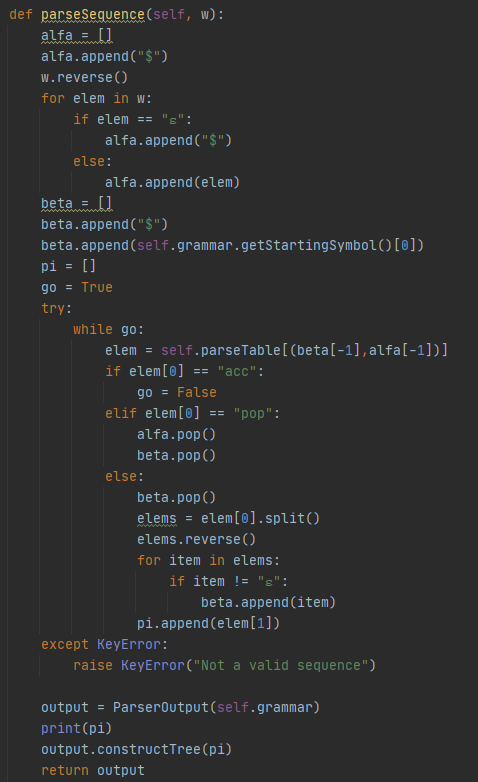
The operation looks like this. Where if we have double epsilon we preserve it otherwise the first or second item will remain in the set depending on if the first item is epsilon or not.



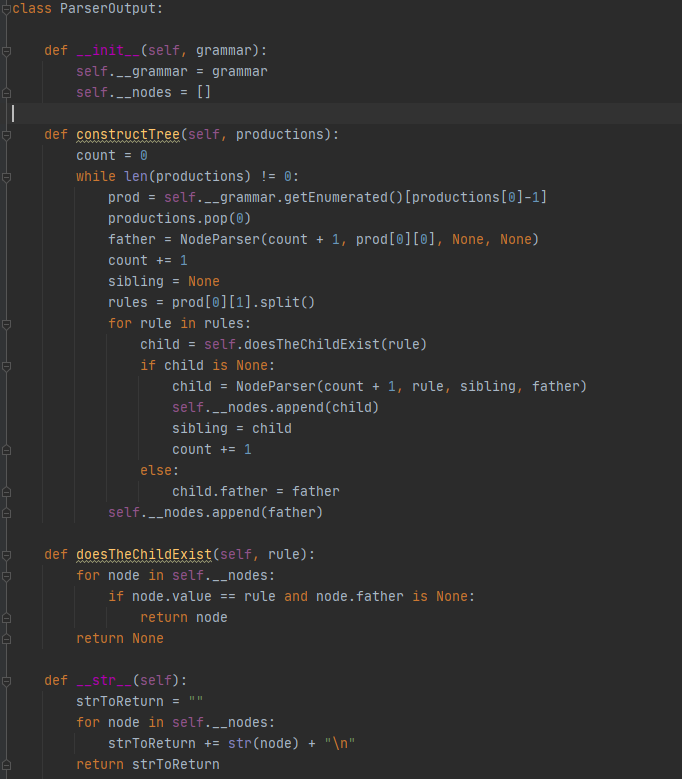
For the follow set creation it is initialized as depicted in the course with the starting symbol having epsilon then the loops represent the iteration and the modified behaves similarly as in the first set. When finding the item on the right hand side we get what is following it and depending on if the first contains epsilon (or it is an empty list which is the same) we will add the first of the that part or the follow of the left hand side



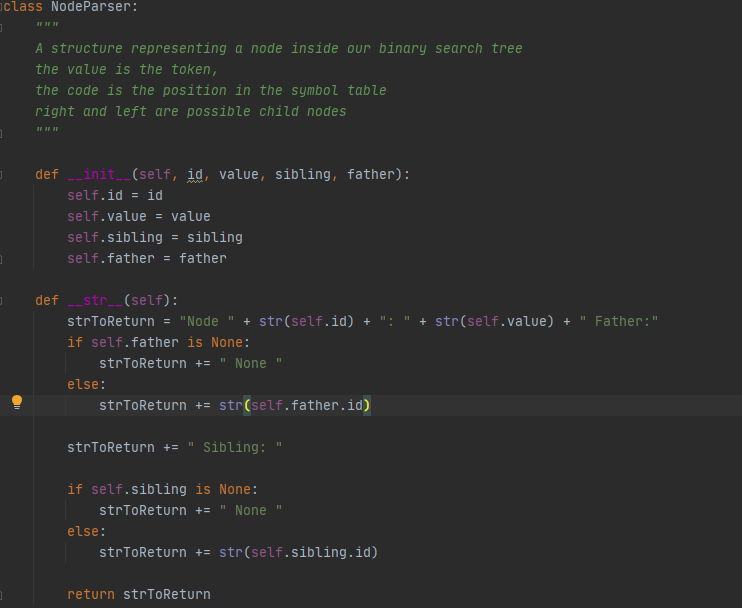
The algorithm which generates the parsing table. The table is generated when the Parser object is instantiated and it uses the follow and first sets to generate the table. It goes through all of the productions and for each production’s left hand side’s first elements are assigned to in the parsing table the right hand side of the production, with the element in the first set as another key, if epsilon appears in the first set the elements that are in the follow set similarly to elements in the first set. Conflicts appear of more than one item is located in a cell of the table in that case the grammar is not ll(1).  
For the terminals and the empty token we assign pop and acc respectively



A sequence is parsed using this algorithm which is the parsing algorithm for ll(1), after initializing the input and working stack for each pair of elements at the end of the working stack we get their production in the parsing table, if it is pop we pop from both of the stacks if acc the sequence is accepted, if we get a production then we pop the working stack and push the elements from the obtained production’s right hand side into it. If we geta KeyError we have an invalid input sequence and an exception is raised.



The parser output is used by the parser to represent the output. In this case we have a tree with the father sibling relation.



The node for the parser output, with a father and a sibling and the current value, each of them is a string.

Documentation for the Scanner and the finite automata can be found in the previous lab documentations.