[Former-languages-and-compiler-design/Lab6 at main · 915-Nistor-Anca/Former-languages-and-compiler-design (github.com)](https://github.com/915-Nistor-Anca/Former-languages-and-compiler-design/tree/main/Lab6)

**Class Grammar:**

*\_\_init\_\_(self, file\_name):* Initializes the Grammar class with the name of the file the grammar is written in, and also with an empty list for the terminals, non-terminals, production rules and the start symbol.

*readFromFile(self):* Reads the grammar from the given file. On the first line, there are the non-terminals, on the second one, the terminals, on the third there is the start symbol and from the fifth line there are the production rules. The function splits the non-terminals and terminals by comma. It adds everything from the file in the Grammar class.

*\_\_str\_\_(self):* Returns a string with evertyhing the class has as attributes, just like it is in the file.

*breakProductionRule(self, production\_rule):* Breaks the given production in two parts, one with the left-hand side (the non-terminal) and the other with what comes after ->.

*checkIfCFG(self):* Checks and returns a message whether the given grammar is a Context Free Grammar or not. Being a CFG means one production rule has to start with the start symbol, the left hand side of each production rule has to be one non-terminal and everything from the right hand side can be found on the non-terminals or terminals list.

**New functions:**

*FIRST(self, symbol):* Computes the FIRST set for a given symbol in the context-free grammar based on specific rules. At first, it initializes an empty set called first\_set to store the computed FIRST set for the symbol.

If the given symbol is a terminal (found in self.terminals), it adds the terminal to the first\_set and returns it. Then, it iterates through the production rules in self.production\_rules to find productions where the left-hand side matches the given symbol.

For each matched production rule: initializes epsilon\_found as True, iterates through the symbols on the right-hand side of the production rule, checks if the current symbol s in the right-hand side is "epsilon". If s is "epsilon", sets epsilon\_found to True. Otherwise, computes the FIRST set (s\_first\_set) for symbol s recursively using the FIRST function. After that, it merges (using the |= operator) the s\_first\_set into the first\_set. If "epsilon" is not in s\_first\_set, sets epsilon\_found to False and breaks the loop. If epsilon\_found is True for all symbols in the right-hand side, it adds "epsilon" to the first\_set.

In the end, it returns the computed first\_set, which represents the FIRST set for the given symbol in the CFG.

*FOLLOW(self, symbol):* Computes the FOLLOW set for a given symbol in the context-free grammar based on specific rules.

At first, it initializes an empty set called follow\_set to store the computed FOLLOW set for the symbol. If the given symbol matches the self.start\_symbol, it adds $ to the follow\_set. Then, it iterates through the production rules in self.production\_rules. For each production rule: splits the production rule into its left-hand and right-hand sides using self.breakProductionRule(p), checks if the given symbol exists in the right-hand side (broken\_p[1]) of the production rule. If the symbol exists in the right-hand side, it retrieves the index (idx) of the symbol in the right-hand side and iterates through the symbols that appear after the symbol in the right-hand side. Then, it computes the FIRST set (first\_s) for each of these symbols (s) using the self.FIRST(s) function. After that, it merges the computed FIRST set into the follow\_set, excluding the epsilon symbol using set difference. If epsilon ("epsilon") is not present in the FIRST set, breaks the loop. If epsilon is present in the FIRST set of all symbols after symbol, it continues the loop. If the loop completes without breaking (i.e., epsilon is in the FIRST set for all symbols after symbol), includes the FOLLOW set of the left-hand side (broken\_p[0]) of the production rule recursively by calling self.FOLLOW(broken\_p[0]) and merges it into the follow\_set.

Finally, it returns the computed follow\_set, which represents the FOLLOW set for the given symbol in the CFG.

*firstSets(self):* Generates FIRST sets for all non-terminals in the grammar.

*followSets(self):* Generates FOLLOW sets for all non-terminals in the grammar.