Github repo: https://github.com/915-Petruta-Razvan/LFTC_Labs

Documentation

Grammar

Details: class used to represent a grammar read from a file.
Methods:

- PrintNonterminalSymbols(): prints the nonterminals of the grammar
- PrintTerminalSymbols(): prints the terminals of the grammar
- PrintAllProductions(): prints the set of productions of the grammar
- CheckCFG(): checks if a grammar is a context free grammar (the left hand side of a production is a nonterminal and the right hand side is a combination of terminal and nonterminal symbols)
- PrintProductionLHSsForANonterminal(nonterminal: string): prints the productions in which the nonterminal appears in the left hand side
- GetProductionLHSsForANonterminal(nonterminal: string): gets the productions in which the nonterminal appears in the left hand side
- PrintProductionRHSsForANonterminal(nonterminal: string): prints the productions in which the nonterminal appears in the right hand side
- GetProductionRHSsForANonterminal(nonterminal: string): gets the productions in which the nonterminal appears in the right hand side
- PrintStartingSymbol(): prints the starting symbol of the grammar
- InitFromFile(): read the grammar from a file

Production

Details: this class is used to represent a production having a left hand side and a right hand side.

LL1Parser

Details: parser implementation for LL(1)
Methods:

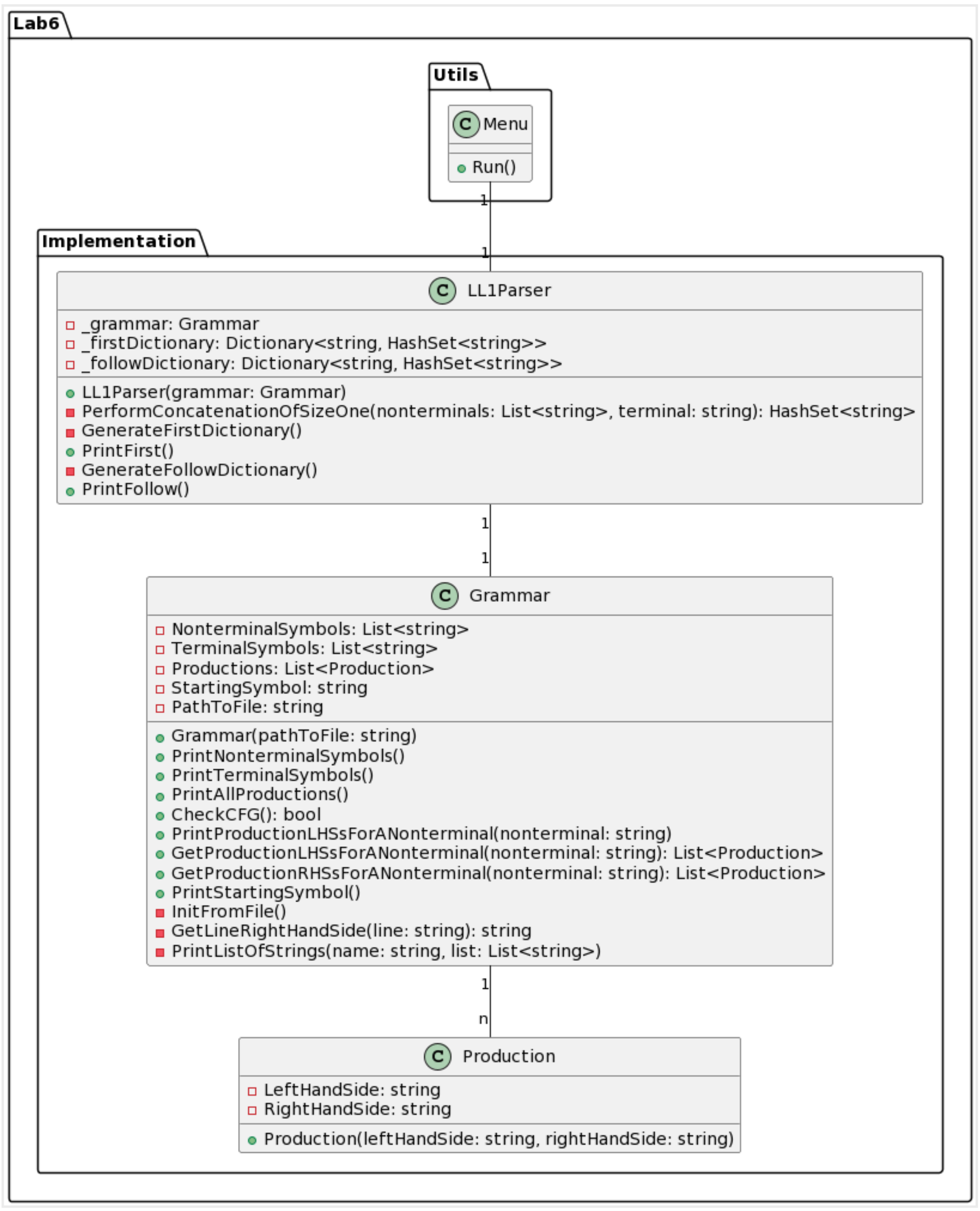
- PerformConcatenationOfSizeOne(nonterminals: List<string>, terminal: string):
    - If a nonterminal can derive ε, the first symbol of a string derived from the sequence could come from the FIRST set of the next nonterminal.
    - The process continues until a nonterminal that cannot derive ε is encountered, or all nonterminals have been checked.
    - if all nonterminals can derive ε, and there's a terminal, the terminal is also included in the resulting set (as the entire nonterminal

    sequence can derive ε, leaving the terminal as the first symbol).
- GenerateFirstDictionary():
    - Initial Pass: The method first adds terminals or ε that are directly derivable from each nonterminal.
    - Iterative Refinement: The method then iteratively refines these sets. This is necessary because the FIRST set of a nonterminal may depend on the FIRST sets of other nonterminals. For example, if a nonterminal A has a production A → B C, then the FIRST set of B (and possibly C, if B can derive ε) contributes to the FIRST set of A.
    - The loop continues until no more changes occur in the FIRST sets
- GenerateFollowDictionary():
    - Initial Setup: The method starts by adding ε to the FOLLOW set of the starting symbol, as it's the first symbol in the derivation process.
    - Handling Productions: The method examines each production where a nonterminal appears on the RHS. Depending on the position and the symbol following the nonterminal, different rules are applied to update the FOLLOW set.
    - Terminal and Nonterminal Handling: If a terminal follows the nonterminal, it's added directly to the FOLLOW set. If another nonterminal follows, the method adds all terminals from its FIRST set (except ε) to the FOLLOW set. If ε is in the FIRST set of this following nonterminal, the FOLLOW set of the LHS nonterminal is also included.
    - Iterative Refinement: The method iteratively updates the FOLLOW sets. Since the FOLLOW set of one nonterminal can depend on others, multiple iterations are needed until no further changes occur.

Menu

Details: this class is used to create a menu for the user to interact with.
Possible options:
- EXIT
- Show nonterminals
- Show terminals
- Show all productions
- Show productions for a given nonterminal (LHS)
- Show productions for a given nonterminal (RHS)
- Show starting symbol
- Is the grammar a context free grammar?
- Show FIRST
- Show FOLLOW

Class Diagram


Lab6
Utils
Menu
+ Run()
1
1
Implementation
LL1Parser
- _grammar: Grammar
- _firstDictionary: Dictionary<string, HashSet<string>>
- _followDictionary: Dictionary<string, HashSet<string>>
+ LL1Parser(grammar: Grammar)
- PerformConcatenationOfSizeOne(nonterminals: List<string>, terminal: string): HashSet<string>
- GenerateFirstDictionary()
+ PrintFirst()
- GenerateFollowDictionary()
+ PrintFollow()
1
1
Grammar
- NonterminalSymbols: List<string>
- TerminalSymbols: List<string>
- Productions: List<Production>
- StartingSymbol: string
- PathToFile: string
+ Grammar(pathToFile: string)
+ PrintNonterminalSymbols()
+ PrintTerminalSymbols()
+ PrintAllProductions()
+ CheckCFG(): bool
+ PrintProductionLHSsForANonterminal(nonterminal: string)
+ GetProductionLHSsForANonterminal(nonterminal: string): List<Production>
+ GetProductionRHSsForANonterminal(nonterminal: string): List<Production>
+ PrintStartingSymbol()
- InitFromFile()
- GetLineRightHandSide(line: string): string
- PrintListOfStrings(name: string, list: List<string>)
1
n
Production
- LeftHandSide: string
- RightHandSide: string
+ Production(leftHandSide: string, rightHandSide: string)

Testing the algorithm:

Grammar from seminar 8:

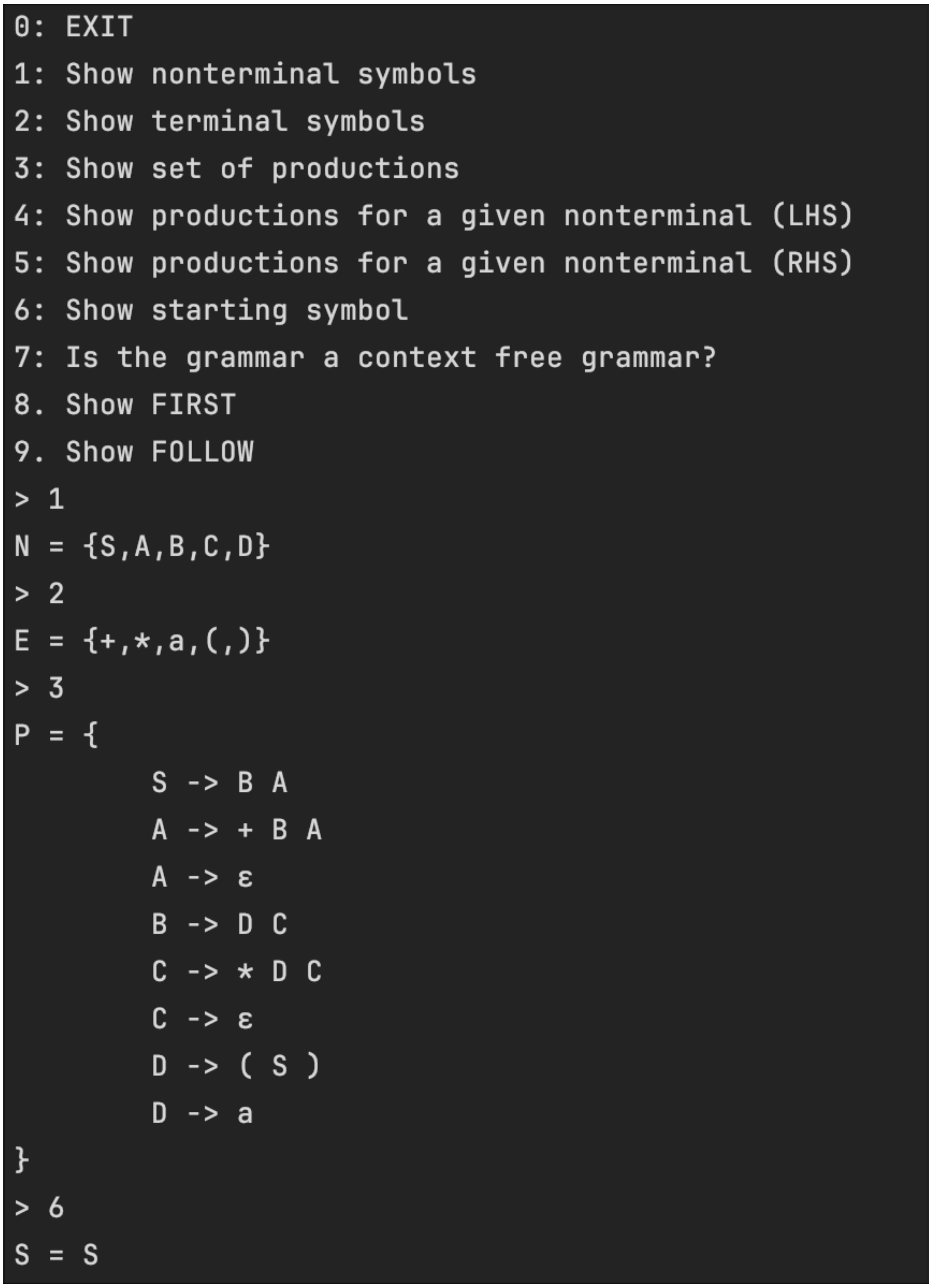
Ex.: Given the CFG $G = (\{S, A, B, C, D\},\ \{+, *, a, (,)\},\ P,\ S)$,

P:
(1) $S \rightarrow BA$
(2) $A \rightarrow\ +BA$
(3) $A \rightarrow \varepsilon$
(4) $B \rightarrow DC$
(5) $C \rightarrow\ *DC$
(6) $C \rightarrow \varepsilon$
(7) $D \rightarrow (S)$
(8) $D \rightarrow a$,

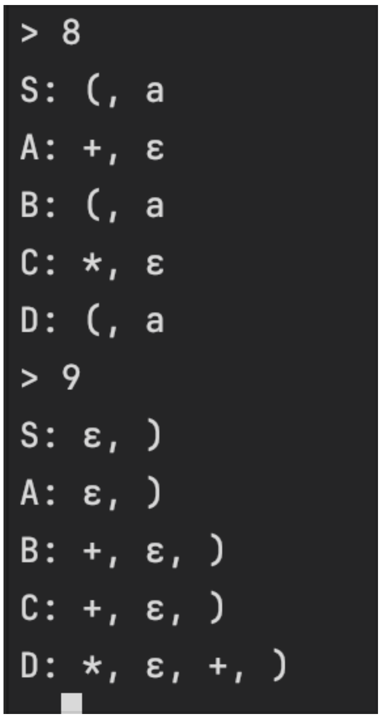
FIRST(S) = { (, a }
FIRST(A) ={+, ε}
FIRST(B) ={ (, a }
FIRST(C) ={*, ε}
FIRST(D) ={ (, a }

FOLLOW(S) = {ε,) }
FOLLOW(A) = {ε,) }
FOLLOW(B) = {+, ε,) }
FOLLOW(C) = {+, ε,) }
FOLLOW(D) = {*,+, ε,) }

Program execution:



```
0: EXIT
1: Show nonterminal symbols
2: Show terminal symbols
3: Show set of productions
4: Show productions for a given nonterminal (LHS)
5: Show productions for a given nonterminal (RHS)
6: Show starting symbol
7: Is the grammar a context free grammar?
8. Show FIRST
9. Show FOLLOW
> 1
N = {S,A,B,C,D}
> 2
E = {+,*,a,(,)}
> 3
P = {
        S -> B A
        A -> + B A
        A -> ε
        B -> D C
        C -> * D C
        C -> ε
        D -> ( S )
        D -> a
}
> 6
S = S
```



```
> 8
S: (, a
A: +, ε
B: (, a
C: *, ε
D: (, a
> 9
S: ε, )
A: ε, )
B: +, ε, )
C: +, ε, )
D: *, ε, +, )
```