LFTC - Lab 6

https://github.com/samzirbo/Formal-Languages-and-Compiler-Design

Class Parser:

The `Parser` class is designed for LL(1) parsing based on a provided context-free grammar. It includes methods for computing the First and Follow sets of non-terminals, constructing a parsing table, and parsing input sequences.

Fields:

- `grammar`: *Grammar* - An instance of the *Grammar* class representing the context-free grammar.
- `table`: *dict* - A dictionary representing the LL(1) parsing table.
- `tree`: *Tree* - An instance of the *Tree* class representing the parse tree.
- `first`: *dict* - A dictionary storing the First sets for each non-terminal.
- `follow`: *dict* - A dictionary storing the Follow sets for each non-terminal.

Methods:

- `isFinal(self, symbol: str) -> bool`: Checks if the First set of a given symbol can be currently computed.
- `First(self) -> None`: Computes the First sets for all non-terminals in the grammar.
- `getFirst(self, symbols: list) -> set`: Computes the First set for a given list of symbols.
- `Follow(self) -> None`: Computes the Follow sets for all non-terminals in the grammar.
- `buildTable(self) -> None`: Computes the First and Follow sets, and builds the LL(1) parsing table.
- `parse(self, sequence: list) -> Tree`: Parses the input sequence using the LL(1) parsing table and outputs the production used, the derivation, and the parse tree.
- `saveTree(self, filename: str) -> None`: Saves the parse tree to a file.
- `printFollow(self) -> None`: Prints the computed Follow sets for each non-terminal.

- `printFirst(self) -> None`: Prints the computed First sets for each non-terminal.
- `printTable(self) -> None`: Prints the LL(1) parsing table as a DataFrame.
- `getPair(self, nonterminal, production) -> Tuple`: Returns the pair (production, productionNo) to be added to the LL(1) parsing table.

First:

- initialize the nonterminals to empty set
- initialize the terminals to themselves
- loop untill all nonterminals their first set have computed
  - check if the first of a specific nonterminal can be computed
  - compute concatenation of length 1 to the first of the rhs of their productions
    - first(S) = the set of the first terminals that can be derived from S

Follow:

- initialize the nonterminals with empty set
- loop until nothings has changed
  - for each nonterminal
    - for each production in which it is found in the rhs
      - if it is the last symbol in rhs
        - add the follow of the rhs to the follow of the nonterminal
      - else, take the first of the following symbols and add to the follow of the nonterminal
        - if epsilon is found in the first, then also add the follow of the lhs