LFTC - Lab 7

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.

Parsing Algorithm:

- check if all symbols in the input sequence are in the terminals of the grammar
- initialize the input and work stack accordingly
- loop until a sequence is rejected or accepted:
  - pop the symbols from the input and work stack
  - take the action corresponding to those symbols
    - if accept or null → break
    - if pop → pop both stacks
    - if push → pop the top of the working stack and push the rhs of the productions
- during parsing, the productions, derivations and the tree structure is build and return as output