LL1 Parser

Requirement:

Implement a parser algorithm for LL1.

Class Grammar (required operations: read a grammar from file, print set of nonterminals, set of terminals, set of productions, productions for a given nonterminal, CFG check).

Algorithms corresponding to parsing table (if needed) and parsing strategy.

Class ParserOutput - DS and operations corresponding to choice 2.a/2.b/2.c (Lab 5) (required operations: transform parsing tree into representation; print DS to screen and to file)

Implementation:

Class Grammar

- Fields:
  - non_terminals – set
  - terminals – set
  - start_symbol – string
  - productions – map key = string (lefthand side of production), value = list (all righthand sides for that lefthand side)
  - is_CFG – bool (True is context free grammar, False otherwise)
- Methods:
  - read_grammar(file_name: string) – read the grammar from file and constructs the grammar, also checking is it's CFG
  - get_productions_string() – return the string with all the productions
  - get_productions_non_terminal(non_terminal : string) – return the array with all the corresponding productions
  - check_CFG() – return the field is_CFG
- Input file:
  - `file ::= non_terminals newline terminals newline start_symbol newline productions`
  - `letter ::= "A" | "B" | … | "Z" | "a" | "b" | … | "z"`
  - `digit ::= "0" | "1" |...| "9"`
  - `newline ::= '\n'`
  - `space ::= " "`
  - `special_characters ::= *all special characters*`
  - `non_terminal ::=  (letter | {letter}) [digits]`
  - `non_terminals ::= {non_terminal space} non_terminal`
  - `terminal ::= ( special_characters | letter | digit) { special_characters | letter | digit  }`

Class Parser

- first_non_terminal(non_terminal: string) – compute the set of starting terminals corresponding to the given non-terminal (take each production where the non-terminal appears); also consider epsilon productions
- first(sequence: string) - compute the set of starting terminals corresponding to the given sequence (that is the right hand side of a production); take into account epsilon productions
- compute_follow_set() – compute the follow set for all the non-terminals in the grammar
- ll1_table() -> dict - creates the ll1 table for the given grammar with the first and follow sets of the symbols
- parsing_algo(sequence: string) -> list| None - takes a string sequence as input and checks if it is accepted by the grammar, if acceted returns the output stack of productions, None otherwise.
- tree(output_stack: list) -> None – takes a output stack and constructs a parsing tree coresponding to the output stack and the productions of the grammar

Github: https://github.com/AdaGabi/Parser