GitHub link: https://github.com/Gabarsolon/FLCD-Parser

Production

```
def __init__(self, left_hand_side, right_hand_side):
    self.left_hand_side = left_hand_side
    self.right_hand_side = right_hand_side
```

Analysis element

```
def __init__(self, production, prefix_position)
 self.production = production
 self.prefix_position = prefix_position
```

Grammar

read_grammar_from_file(*self*, file_path): Reads grammar from file. The non-terminals are on the first line, the terminals on the second, the start symbol on the third and the rest of the lines are the productions.

productions_for_a_given_non_terminal(self, non_terminal): Returns the productions of a given non-terminal if it's valid.

cfg_check(*self*): Checks if all the productions come from a non-terminal

closure(*self*, analysis_element): Function that gets as a parameter a set of analysis elements and it returns another set of analysis elements containing the current set concatenated with all productions that are after the dot for a non terminal

goto(*self*, analysisElements, symbol): Function that finds all the analysis elements from a state which have . right before a given symbol, moves that . after the symbol and calls the closure function on it

canonicalCollection(*self*): Function that builds the canonical collection based on a new starting symbol

get_production_number(self): Function that gets the index of the production in the list of all productions

get_all_productions_separated(self): Function that transforms the dictionary of productions into a list

get_producition_given_his_number(self): Function that returns a production given his index in the list of all productions

parsing_table(self): Function that returns the parsing table, represented in the following way:

```
(State 0 is in the position 0 of the list)
[
  {
    ACTION: ["shift" | "accept" | "r1" | "error,]
    GOTO: {
      a: 2,
      A: 3
      ...
    }
  }
]
```

parse_sequence(self, sequence): function that gets a sequence and checks whether the sequence is valid or not by using the parsing table and returns the output band or None, if it is not valid

```
class Entry:
  def __init__(self, index, info):
    self.index = index
    self.info = info
    self.parent = None
    self.right_sibling = None
    self.left_child = None

class ParserOutput:
  def __init__(self, grammar):
    self.current_index = 1
    self.root = Entry(self.current_index, "")
    self.grammar = grammar
    self.indexInput = 1
```

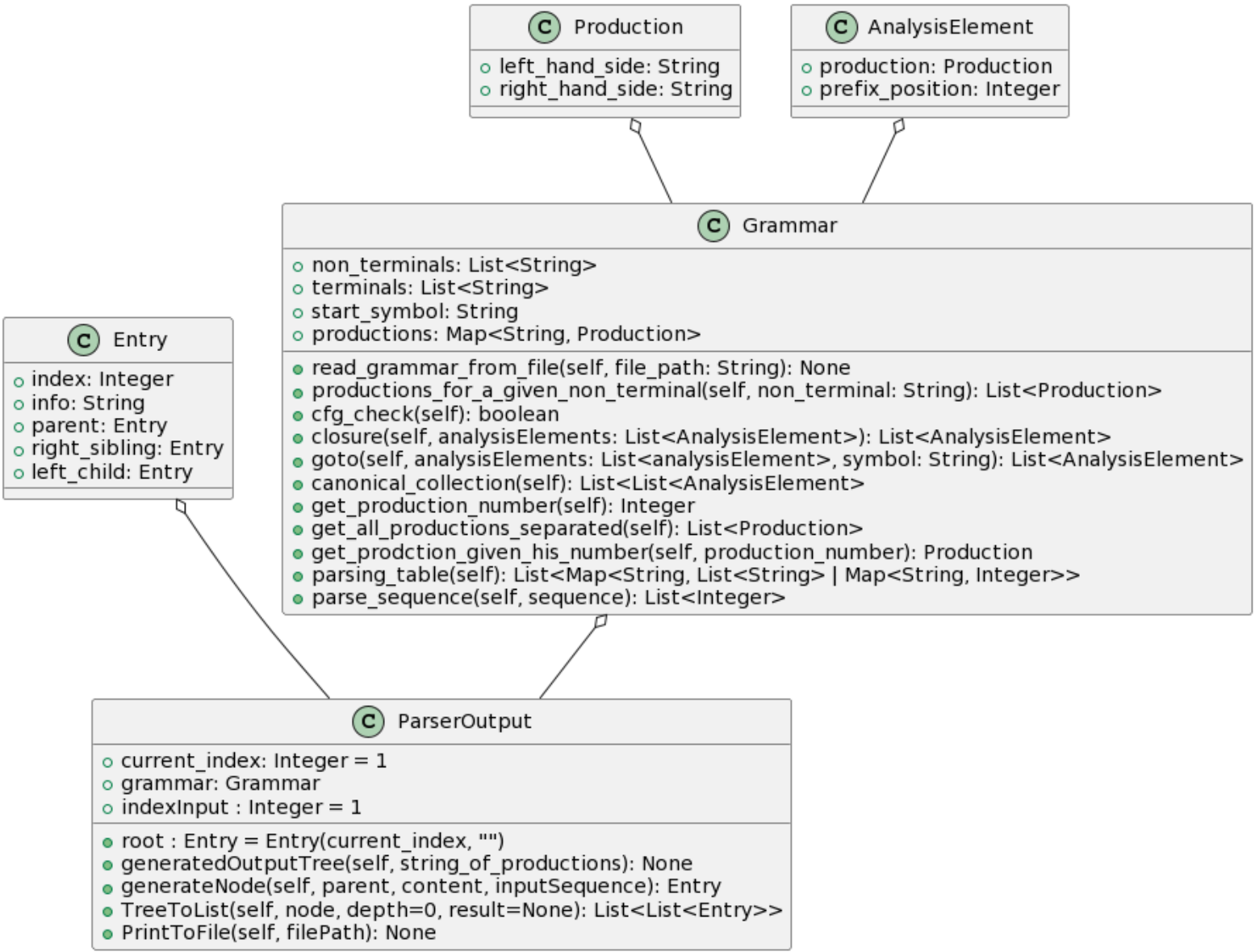
generateOutputTree(self, string_of_productions): Function that generates the father and sibling relation tree from a given string of productions

generateNode(self, parent, content, inputSequence): Recursive function that generates the nodes for the tree

TreeToList(self, node, depth=0, result=None): Function that transforms the tree into a list that looks similar to this:

```
[{'index': 1, 'info': 'S', 'parent': None, 'right_sibling': None}]
[{'index': 2, 'info': 'a', 'parent': 'S', 'right_sibling': 'A'}, {'index': 3, 'info': 'A', 'parent': 'S', 'right_sibling': None}]
[{'index': 4, 'info': 'b', 'parent': 'A', 'right_sibling': 'A'}, {'index': 5, 'info': 'A', 'parent': 'A', 'right_sibling': None}]
[{'index': 6, 'info': 'b', 'parent': 'A', 'right_sibling': 'A'}, {'index': 7, 'info': 'A', 'parent': 'A', 'right_sibling': None}]
[{'index': 8, 'info': 'c', 'parent': 'A', 'right_sibling': None}]
```

PrintToFile(self, filePath): Function that creates a file (within the given path) which contains the output of the TreeToList function


Production
left_hand_side: String
right_hand_side: String
AnalysisElement
production: Production
prefix_position: Integer
Grammar
non_terminals: List<String>
terminals: List<String>
start_symbol: String
productions: Map<String, Production>
read_grammar_from_file(self, file_path: String): None
productions_for_a_given_non_terminal(self, non_terminal: String): List<Production>
cfg_check(self): boolean
closure(self, analysisElements: List<AnalysisElement>): List<AnalysisElement>
goto(self, analysisElements: List<analysisElement>, symbol: String): List<AnalysisElement>
canonical_collection(self): List<List<AnalysisElement>
get_production_number(self): Integer
get_all_productions_separated(self): List<Production>
get_prodction_given_his_number(self, production_number): Production
parsing_table(self): List<Map<String, List<String> | Map<String, Integer>>
parse_sequence(self, sequence): List<Integer>
Entry
index: Integer
info: String
parent: Entry
right_sibling: Entry
left_child: Entry
ParserOutput
current_index: Integer = 1
grammar: Grammar
indexInput : Integer = 1
root : Entry = Entry(current_index, "")
generatedOutputTree(self, string_of_productions): None
generateNode(self, parent, content, inputSequence): Entry
TreeToList(self, node, depth=0, result=None): List<List<Entry>>
PrintToFile(self, filePath): None