github: https://github.com/AndrewBarzu/LFTCParser

Classes:

RecursiveDescendantParser:

- constructor
  - input -> grammar: Grammar (from last lab)
- parse
  - input -> word: Tuple[str] – the word to be parsed
  - output -> [] if sequence is not accepted, list with derivation graph is sequence is accepted
  - side-effect -> Prints 'Error' if sequence is not accepted, and 'Sequence accepted' if it is accepted

RecursiveDescendantConfiguration:

- constructor
  - input -> grammar: Grammar
  - side-effect -> sets the current state to q, current index (i) to 0, working stack to empty list and input stack to a list containing only the starting symbol
- expand
  - input -> None
  - output -> None
  - side-effect -> pops the current nonterminal from the input stack, annotates it with $0 (marking the 1st production) and pushes it onto the working stack. It also pushes into the working stack the 1st production of the current nonterminal
- advance
  - input -> None
  - output -> None
  - side-effect -> increments i, pops the current terminal from the input stack and pushes it onto the working stack
- momentaryInsuccess
  - input -> None
  - output -> None
  - side-effect -> sets the current state to b
- back
  - input -> None
  - output -> None
  - side-effect -> decrements i, pops the current terminal from the working stack, and then it pushes it into the input stack
- anotherTry

    - input -> None
    - output -> None
    - pops the current annotated nonterminal from the working stack, gets the next production for that nonterminal and pushes it into the input stack, while also increasing the annotated value and pushing the new annotated nonterminal onto the working stack if the current nonterminal has a next production. If the production does not have a next production, then it just appends the nonterminal without the annotation back onto the input stack, or if i is 0 and the nonterminal is the starting symbol of the grammar, then it sets the state to e
- success
    - input -> None
    - output -> None
    - side-effect -> sets the current state to f

The derivation graph returned by the parsers's `parse` method is an ParserOutput object, containing a list L of the structure:

- $L = [$
  $e \in L \mid e = (\text{information}, \text{parent}, \text{sibling}),$
  $\text{information} = \text{grammar symbol},$
  $\text{parent} = \{i \mid L[i][\text{information}] = \text{parent of e in the parsing algorithm}\},$
  $\text{sibling} = \{i \mid L[i][\text{information}] = \text{right sibling of e in the parsing algorithm}\}$
  $]$

ParserOutput:

- constructor
    - input
        - grammar: Grammar
        - productionString: the production string resulted from the recursive descendant parser (the final working stack)
- plotParseTree
    - input
        - filename: string
    - output
        - inside the file <filename>, the parse tree will be plotted.