Team members:
Condrea Adrian
Deiac David

FLCD Parser - Documentation

Parsing method: **recursive descendent**
Representation of the parsing tree (output): **derivations string** (max grade = 9)
Programming language: **Python**
Github link: https://github.com/adriancondrea/FLCD_Parser

Structure of the project:
The root directory contains three subdirectories:
- **docs** (documentation)
- **input** (with **grammars** and input **sequences** subdirectories)
- **output** with the result of running the parser on the given grammar with the given input sequence

Grammar class:

The grammar class stores the list of non-terminals N, the list of terminals E, a dictionary P for the production rules and the starting symbol S.

check_valid_cfg method checks that the starting symbol is a non-terminal, and for each production, checks that the left hand side is a non-terminal and the right hand side is a collection of terminals and/or non-terminals, or epsilon.

Parser class:

The parser class stores the starting symbol, the grammar, the state of parsing (s), the position of current symbol in input sequence (i), the working stack, which stores the way the parse is built, the input stack, which is part of the tree to be built, the input string and its length, along with a boolean flag accepted, which represents whether the input string is accepted by the grammar.

The parser class implements the parsing methods: expand, advance, momentary insuccess, back, another try and success.

The parsing algorithm loops while the state is not final and there are no errors. If the state is ‘q’ (normal state), we check if the position of the current symbol in input sequence is greater than the input string with 1, and call the success method. Otherwise, we check the first symbol in the input stack. If it is a non-terminal we expand, if it is a terminal equal to the current element in the input string we advance, otherwise we call momentary insuccess. If the state is back and the top of the working stack is a terminal, we perform back. Otherwise (the top of the working stack is a non-terminal) we perform another try. If the state is e (error), we print an error message. Otherwise, the input sequence was accepted, we print a message and set the flag.

Parser output class:

The parser output class gets as argument the parser and computes the derivation list.

The compute derivation string method takes the working stack of the parser, goes to each entry, and computes the derivations string.