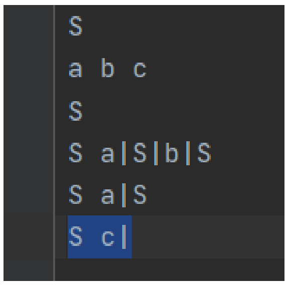
Recursive Descendent Parser

GitHub Link: https://github.com/pauladam2001/Sem5_FormalLanguagesAndCompilerDesign/tree/master/Lab5-Python

Simple example:

G1.txt:



```
S
a b c
S
S a|S|b|S
S a|S
S c|
```

Productions are split by the symbol “|”.

The productions for non-terminal S will be:

S -> a S b S

S -> a S

S -> c

Grammar class:

Properties:

1. Filename -> string;
2. Grammar -> list of lists (representation of objects);
3. N -> list of non terminals;
4. E -> list of terminals;
5. S -> starting symbol (as a list of one string);
6. P -> productions kept as a dictionary with a string as key and value a list, that contains lists of symbols in the right hand side of the production.

Methods:

1. read_grammar() -> will read grammar from text file;

2. represent_productions() -> will construct the P dictionary;
3. get_non_terminals();
4. get_terminals();
5. get_start_symbol();
6. get_productions();
7. get_productions_for_non_terminal();
8. print_productions_for_non_terminal().

Parser class:

Properties:

1. grammar -> Grammar;
2. sequence -> list of codes;
3. output_file -> string;
4. input_stack -> list acting as a stack;
5. working_stack -> list acting as a stack;
6. state -> string;
7. index -> int;
8. tree -> list.

Methods:

1. read_sequence() -> will build the list of codes from the file;
2. write_all_data() -> will append to the file the current state, index, the content of the working stack and the content of the input stack;
3. init_output_file() -> will create the output file;
4. write_in_output_file() -> will append a message in the output file;
5. expand();
6. advance();
7. momentary_insuccess();
8. back();
9. success();
10. another_try();
11. print_working() -> will print the working stack and append it to the output file;
12. run() -> the main function, will check if the sequence is accepted;
13. create_parsing_tree();

14. get_length_depth();
15. write_parsing_tree().

Algorithm:

We have an initial configuration and we define some moves to get to the final configuration. A configuration is of the model (s, i, α, β) where s is the state of the parsing, i is the position of current symbol in the input sequence, α is the working stack that stores the way the parse is built and β is the input stack, that is part of the tree to be built.

The state s can be:

1. n-> normal state;
2. b -> back state;
3. f -> finals state that signifies success;
4. e -> error state that signifies insuccess.

The moves can be:

1. Expand -> when the head of input stack is nonterminal. We pop the nonterminal from the input stack, then we put it to the top of the working stack and then we put to the top of the input stack a new production for the nonterminal.
2. Advance -> when the head of input stack is a terminal = current symbol from input, we pop the terminal from the input stack and put it at the top of the working stack. After that we increment the index.
3. Momentary insuccess -> when the head of the input stack is a terminal != current symbol from input. We set the current state to the back state “b”.
4. Back -> when the head of working stack is a terminal. We pop the terminal from the working stack and put it at the top of the input stack. After that we decrement the index.
5. Another try -> when the head of the working stack is a nonterminal. We pop the nonterminal from the working stack and then we have three cases. We have to keep in mind that an element in the working stack is a tuple of the form (nonterminal, production number).
   a) If the production number + 1 is smaller than the number of productions for the nonterminal, we set the state to “n”, we construct the new tuple that consists of the nonterminal and the

new production number and put it on the top of the stack. After that, we delete the production at the end of the input stack and we put the new production at the top of the input stack.

   b) If the index is 1 and the nonterminal is the starting symbol, we set the state to “e”;
   c) Otherwise we delete the production at the end of the input stack and we put the nonterminal at the top of the input stack.

6. Success -> we set the state to “f”.

Parsing Tree:

The parsing tree is represented as a table of the form (index, info, parent, left_sibling).