Parsing using LL(1)

Start by using the same functions for reading a grammar from the previous laboratory.
I have the *get_set, string_to_set, string_to_map and read_grammar* functions.
Now I need to compute the FIRST of a non-terminal.
To make my work easier I also created the function that changes the productions string into a hash that has as value a list of lists. Each value contains a list with all the possible productions and each production - list from the big list - is composed of the elements of a production. e.g: S -> AB | c+E | epsilon; will be translated into this hash: {'S': [['A', 'B'], ['c', '+', 'E'], []]}.

First

Now I compute FIRST by creating a dictionary which has as keys:

- the terminals. The values for this keys are the same with the keys.
- the non terminals.

The rules for computing FIRST are as following:
In a first iteration, F0, add in the set all the terminals that are on a first position in a production or epsilon. Then, repeatedly do this until all Fi are the same sets as Fi-1:

- Concatenate all the FIRST sets from the previous steps of the RHS of each production and take the first elements. If some empty set occurs at a time, avoid the process for that production at that step. If all of them have epsilon, add epsilon.

Follow

Now I compute FOLLOW with the following rules:

1. FOLLOW(S) = { $ }   // where S is the starting Non-Terminal
2. If A -> pBq is a production, where p, B and q are any grammar symbols,
   then everything in FIRST(q) except Є is in FOLLOW(B).

3. If A->pB is a production, then everything in FOLLOW(A) is in FOLLOW(B).

4. If A->pBq is a production and FIRST(q) contains Є,
   then FOLLOW(B) contains { FIRST(q) – Є } U FOLLOW(A)

- Take all the occurences of some non terminal B in some production and proceed with the steps from above

Parsing table

Computing the parsing table means iterating all the productions and for each production that has epsilon in the FIRST set of the RHS, add that production in the corresponding terminals from the FOLLOW of that symbol, otherwise, add the production in the terminals from the

FIRST set of the RHS. If more than 1 entry is in one cell, throw err. For each terminal, write “pop” on their row and column and for $ write “acc”.

Syntactic Analysis

Initialize input stack with the input sequence, work stack with the starting symbol and the output array as an empty array.

As long as the sequence is accepted or denied, do the following steps:

- if the top of the stacks are the same, POP
- otherwise, pop from the work stack as W, add from the parsing table into the work stack the element from the cell on the row corresponding to W and the column corresponding to the terminal form the top of the input stack. Also, add to the output array the index of the corresponding production.
- if the work stack is empty
    - if the input stack is empty -> sequence accepted
    - else -> sequence not accepted
- if the input stack is empty
    - while the non terminal from the top of the work stack goes into epsilon with some terminal, pop it from the work stack and add to the output array the index of the corresponding production
    - if the work stack is empty after this process -> sequence accepted
    - else -> sequence not accepted