Luca Denisa & Lascău Bogdan

https://github.com/hiImDeni/flcd/tree/master/lab5-parser

Lab 5

Assignment for a team of 2 students!

Statement: Implement a parser algorithm

One of the following parsing methods will be chosen (assigned by teaching staff):

1.a. recursive descendent

1.b. ll(1)

1.c. lr(0)

The representation of the parsing tree (output) will be (decided by the team):

2.a. productions string (max grade = 8.5)

2.b. derivations string (max grade = 9)

2.c. table (using father and sibling relation) (max grade = 10)

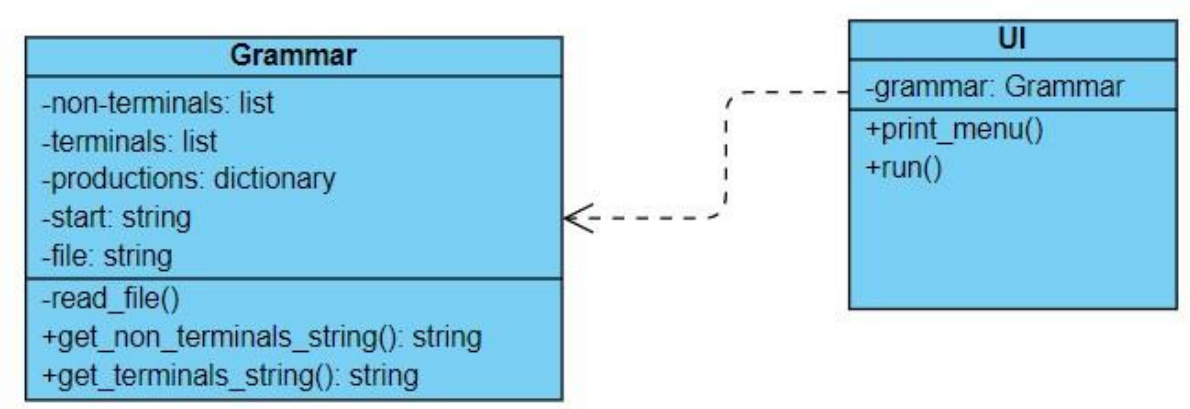
PART 1: Deliverables

1. Class grammar (required operations: read a grammar from file, print set of nonterminals, set of terminals, set of productions, production for a given nonterminal)
2. Input file: g1.txt (grammar from seminar); g2.txt (grammar of the minilanguage; syntax rules from Lab1)
3. Functions corresponding to parsing strategy (see table below)

IMPORTANT remark: conflicts need to be solved, so apply functions for g2.txt

Recursive Descendent	LL(1)	LR parsers
Functions corresponding to moves (expand, advance, momentary insucces, back, another try, succes)	Functions FIRST, FOLLOW	Functions Closure, goto, CanonicalCollection

Implementation



The grammar will read its values from a file (called g1.txt in our case) which is shown below:

```
Z S A
a b c
Z-S
S-a A
A-b A|c
```

The first line represents the non-terminals, which we save in a list which we check for duplicates.

The second line represents the terminals, saved in the same manner as the non-terminals.

The third line and further represent the productions, which we save in a dictionary that has terminals as keys and lists as values. Here, the elements that begin the line represent the nonterminal.

The start symbol will be initialized with the first symbol from the non-terminal list.

closure(element: (string, list))

- element: tuple of the form (symbol, list of productions)
- computes the closure for symbol
- return: dictionary containing symbols as keys and a list of productions as values
- until there is no new production
    - for every symbol in the list of productions
        - check if the given symbol is a non-terminal
        - if it is, add its production to the return dictionary

goto(state: dictionary, symbol: string)

- return: dictionary containing symbols as keys and a list of productions as values

- for the the given state, we check if the symbol is preceded by dot, we move the dot on the next position and call closure on the elements following dot

- if the the is at the end of the production, we return the element unchanged

ColCan()

- we initialize the productions, placing dot at the beginning of the right-hand side
- for every state and for every symbol of the grammar, we call goto and add the result to states, until there is no new state to add
- return: list of states

action(state: dictionary)

- determines what action to be taken for a given state
- return: string or, in case of “reduce”, tuple of form: (action, production_index)

buildTable(states: list of dictionaries)

- for every state, we add to the table its index, the action taken, and the values of the goto function
- return: matrix with 3 columns -> 0 = state index; 1 = action; 2 = dictionary containing goto results for every symbol in the state

parse(word: string)

- build the canonical collection using ColCan
- check if the given word is accepted by the grammar