Documentatie LFDC

Opris Vlad Marius 935, Socaciu Mihai 938

**Github link:** <https://github.com/Socca98/LFTC-2020>

Lab5 – Parser LR(0)

**Requirements**

Class Grammar reading from file.

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 non-terminals, set of terminals, set of productions, production for a given nonterminal)
2. Input file: g1.txt (grammar from seminar); g2.txt (grammar of the mini-language; syntax rules from Lab1)
3. Functions corresponding to parsing strategy (see table below)

Grammar.py

Used by parsing algorithm to construct the syntax tree.

**Properties:**

N – list, represents non-terminals

E – list, represents terminals

S – list, represents starting symbol

P – list, represents productions

Methods:

**read\_from\_file**(filename): read the elements from a file

in: filename – string

out: Grammar object

**parse\_line**(line): parses a line based on comma and spaces

in: line – string

out: list

**parse\_productions**(line): parses a line and forms productions

in: line – string

out: list of productions

**get\_productions**(symbol): get productions for a certain symbol

in: symbol – string

out: production

Parser.py

LR(0) parsing algorithm bottom-up to construct the syntax tree.

**Properties**:

grammar – the Grammar

workingStack – stack

inputStack – stack

output – stack

Methods:

**closure**(productions):

Takes a state containing productions.

In: productions – List of productions for closure

Out: closure

**go\_to**(state, symbol):

Transition from a state to another using a terminal or non-terminal.

in: state – String

symbol – String

out: a closure of a list

**canonical\_collection**():

Construct a set of states.

C - canonical collection

ex: [('S1', ['.', 'program']), ]

out: Collection of states

**generate\_table**(state, symbol):

Generates the parsing table used to check the input tokens.

out: parsing table

**parse**(input\_string):

Using the parsing table we check the input stack if is syntactically correct.

in: inputStack obtained from PIF

out:

UML diagram bellow

