https://github.com/alexandra-olahut/FLCD-assignments

FLCD Documentation

Olahut Alexandra ~ 935/2 ~ Muresan Cristian

Parser LL(1)

Grammar

Fields:

- *nonterminals*: list of nonterminals
- *terminals*: list of terminals
- *start_symbol*: the starting nonterminal
- *productions*: dictionary where each key is a left hand side and the value is a list of tuples, each containing the right hand side and the index of the production; right hand side is represented as a list of symbols
  (e.g: S->aB | ε becomes (S)->[(['a','B'], 1), (['eps'], 2)])

  * epsilon is represented and used as 'eps'

Methods:

- *read_from_file(file_path):* initialized the grammar from a given file
- *isCFG()*: checks if grammar is cfg
- *getters* for all fields

Parser

Fields:

- *grammar*
- *FIRST*, *FOLLOW*: dictionaries where keys are nonterminals and the values are sets
- *table*: dictionaries where the keys are pairs (terminal, terminal) / (terminal, nonterminal), and the values are the values corresponding in the parser table, as tuples of (right hand side, production index) / pop / acc

Methods:

- *buildFirst(), buildFollow, buildTable:* initialize the fields
- *parse(sequence):* returns the productions string if the sequence is accepted by the grammar, empty list and error otherwise

Helper methods:

- *concatenation1(l1,l2):* concatenation of length 1 for 2 given sets
- *concatenateAll(first, rhs):* with a given first function, for a given list of symbols, calculates the first element for the concatenated sequence
- *isCalculated(first, rhs):* with a given first function, checks if for a given right hand side, first for all elements was calculated

ParserTree

Fields:

- *grammar*
- *nodes*: list of nodes

Node – represented with an index, information, parent, left sibling and a Boolean specifying if the node had been processed or not

Methods:

- *getProduction(index):* returns lhs, rhs for the production with the given index
- *buildTree(productions)*: builds the parse tree for a given productions string

UI

The user can:

- read a grammar from a file (this also initializes the parser for the grammar: table etc)
- see properties (grammar elements, first, follow, parsing table)
- parse a sequence from given file and output the resulting tree in a given file