https://github.com/RazvanAndreiMoga/LFTC

State enum – it contains all possible states a configuration can receive during the execution of the

parser algorithm:

a) NORMAL_STATE – it is found in success, expand, advance and momentary insuccess moves;

b) BACK_STATE – it is found in back and another try moves;

c) FINAL_STATE – it is received in the end, when the algorithm is successfully finished;

d) ERROR_STATE – it is received when something wrong is happening during the execution of the parser algorithm (e.g.: when the given sequence cannot be parsed)

Move enum – it contains all possible moves that the parser algorithm can assign:

a) SUCCESS – accessed when the state of the parsing is normal, the sequence is entirely processed and the input stack is empty;

b) EXPAND – accessed when the state of parsing is normal and the top of the input state is a nonterminal;

c) ADVANCE – accessed when the state of parsing is normal and the top of the input state is a terminal which is equal to the current element in the sequence (if it exists);

d) MOMENTARY INSUCCESS – accessed when the state of parsing is normal and the top of the input state is a terminal which is not equal to the current element in the sequence;

e) BACK – accessed when the state of parsing is back and the top of the working stack is a terminal;

f) ANOTHER TRY - accessed when the state of parsing is back and the top of the working stack is a nonterminal;

```java
/**
 * This function implements the logic of EXPAND move: creates the next configuration of
 the algorithm, removing the nonterminal from the top of the input stack and processing the
 production indicated by the count parameter - adds the nonterminal to the working stack to
 know which one is used and its corresponding production to the input stack.
 * @param configuration : the current configuration (which will be changed)
 * @param count : an integer number representing the number of the production to be
processed for the current nonterminal
 * @param grammar : the grammar of the language (the one from the fifth laboratory)
 * @return : the new (modified) configuration
 */
1 usage
public Configuration expand(Configuration configuration, int count, Grammar grammar) {
```

```java
/**
 * This function implements the logic of ADVANCE move: creates the next configuration of
 the algorithm, removing the terminal from the top of the input stack and adding it to the
 working stack. In the case the removed terminal is epsilon, it will not be added to the working
 stack (because it will never be equal to any element from the sequence).
 * @param configuration : the current configuration (which will be changed)
 * @return : the new (modified) configuration
 */
1 usage
public Configuration advance(Configuration configuration) {
```

```java
/**
 * This function implements the logic of MOMENTARY INSUCCESS move: creates the next
 configuration of the algorithm, changing the state of parsing to BACK_STATE.
 * @param configuration : the current configuration (which will be changed)
 * @return : the new (modified) configuration
 */
1 usage
public Configuration momentaryInsuccess(Configuration configuration) {
    configuration.setMove(Move.MOMENTARY_INSUCCESS);
    configuration.setStateOfParsing(State.BACK_STATE);
    return configuration;
}

/**
 * This function implements the logic of BACK move: creates the next configuration of the
 algorithm, removing the terminal from the top of the working stack and adding it to the input
 stack (it decreases the position of the current symbol table because we go back with the process
 of searching for a position).
 * @param configuration : the current configuration (which will be changed)
 * @return : the new (modified) configuration
 */
1 usage
public Configuration back(Configuration configuration) {
```

```java
/**
 * This function implements the logic of ANOTHER TRY move: creates the next configuration
 of the algorithm, trying another production for the nonterminal from the top of the working
 stack if it exists or adding the nonterminal back to the input stack otherwise. In case we return
 to the start symbol, it is an error.
 * @param configuration : the current configuration (which will be changed)
 * @param grammar : the grammar of the language (the one from the fifth laboratory)
 * @return : the new (modified) configuration
 */
1 usage
public Configuration anotherTry(Configuration configuration, Grammar grammar) {...}

2 usages
private void anotherTry_helper(Configuration configuration, Grammar grammar, String nonTerminal, in

/**
 * This function implements the logic of SUCCESS move: creates the next configuration of
 the algorithm, changing the state of parsing to FINAL_STATE.
 * @param configuration : the current configuration (which will be changed)
 * @return : the new (modified) configuration
 */
1 usage
public Configuration success(Configuration configuration) {
```

```java
    * This function implements the parser algorithm, calling the above described functions
    when they meet their conditions. This function uses the following
    methods:
    -> private boolean verifyAdvance(String[] sequence, Grammar grammar,
    Configuration configuration) - contains all conditions needed to access advance move;
    -> public boolean isIdentifier(String identifier) - verifies if the given string is an
    identifier;
    -> public boolean isConstant(String constant) - verifies if the given string is an
    identifier;
    -> private void constructWorkingAndInputStacks(List<Configuration>
    configurations, Configuration configuration)
    * @param sequence : the sequence given by user
    * @param grammar : the grammar of the language (the one from the fifth laboratory)
    */

  1 usage
  public void descendantRecursiveParserAlgorithm(String[] sequence, Grammar grammar) {...}

  /**
   * This function writes to a file the content of a java List.
   * @param path : the location of the file
   * @param configurations : the list of configurations
   */
  2 usages
  public void writeToFile(String path, List<Configuration> configurations) {...}
```

```java
/**
 * This function implements the parser algorithm, calling the above described functions
 when they meet their conditions. This function uses the following
 methods:
 -> private boolean verifyAdvance(String[] sequence, Grammar grammar,
 Configuration configuration) - contains all conditions needed to access advance move;
 -> public boolean isIdentifier(String identifier) - verifies if the given string is an
 identifier;
 -> public boolean isConstant(String constant) - verifies if the given string is a
 constant;
 -> private void constructWorkingAndInputStacks(List<Configuration>
 configurations, Configuration configuration)
 * @param sequence : the sequence given by user
 * @param grammar : the grammar of the language (the one from the fifth laboratory)
 */
1 usage
public void descendantRecursiveParserAlgorithm(String[] sequence, Grammar grammar) {
```