https://github.com/alexreitler/FLCD_Lab5/tree/FinalLabParser

The RecursiveDescendant class implements a recursive descent parser, a top-down parsing technique commonly used for parsing context-free grammars. It facilitates parsing a sequence of tokens based on a given grammar and produces a parser output.

Constructor

• RecursiveDescendant(grammar: Grammar): Initializes the parser with a provided grammar. The initial configuration is set up with an epsilon symbol on both the working stack and input stack.

Public Methods

1. run(w: List<String>): List<String>:

o Parameters: w - List of input tokens.

o Returns: List of strings representing the working stack after parsing.

o Description: Runs the recursive descent parsing algorithm on the input sequence w. It processes the input based on the grammar and updates the parser configuration accordingly.

2. expand():

o Description: Expands a nonterminal on the input stack by replacing it with the corresponding production rule from the grammar.

3. advance():

o Description: Advances the parser position and pushes the current token from the input stack to the working stack.

4. momentaryInsuccess():

o Description: Sets the parser state to indicate momentary insuccess, triggering backtracking.

5. back():

o Description: Backtracks by moving the parser position back and restoring the previous state from the working stack to the input stack.

6. anotherTry():

o Description: Handles error recovery by considering alternative production rules for the current nonterminal and adjusting the parser state and stacks accordingly.

7. success():

o Description: Marks successful completion of parsing by setting the parser
state to "f" (finished).

The Configuration class represents the state of the parser during its execution. It includes information such as the parser state, position in the input sequence, and the contents of the working and input stacks.

Constructor

• Configuration(state: String, position: Int, workingStack:
Stack<String>, inputStack: Stack<String>): Initializes a parser configuration
with the provided state, position, working stack, and input stack.

Properties

• state: String: Represents the current state of the parser ("q" for processing, "b" for backtracking, "e" for end, and "f" for finished).

• position: Int: Represents the current position in the input sequence.

• workingStack: Stack<String>: Represents the working stack of the parser.

• inputStack: Stack<String>: Represents the input stack of the parser.

Methods

• toString(): String: Overrides the toString method to provide a string
representation of the configuration for debugging purposes.

The Output class is responsible for constructing and representing the parse tree structure based on the output of a parser. It facilitates the visualization of the hierarchical relationships between nodes in the parse tree. The class utilizes lists and maps to store information about nodes, their parent-child relationships, and the number of children for each node type.

Constructor

• Output(grammar: Grammar): Initializes an Output instance with a given grammar.

The constructor sets up data structures to store information about nodes, parent-child relationships, and the number of children for nonterminals, terminals, and the epsilon symbol.

Properties

• grammar: Grammar: The grammar associated with the parser output.

• nrChildren: MutableMap<String, Int>: A map storing the number of children for each nonterminal, terminal, or epsilon symbol.

• values: MutableList<String>: A list storing the values of nodes in the parse tree.

• father: MutableList<Int>: A list storing the indices of parent nodes.

• leftChild: MutableList<Int>: A list storing the indices of left children.

• rightSibling: MutableList<Int>: A list storing the indices of right siblings.

Public Methods

1. addProductionString(productionString: List<String>):

o Parameters: productionString - A list of strings representing a production sequence.

o Description: Adds a production sequence to the parse tree structure.

2. add(node: String, parent: Int): Int:

o Parameters:

▪ node - A string representing a node in the parse tree.

▪ parent - An integer representing the index of the parent node in the parse tree.

o Returns: The index of the added node in the values list.

o Description: Adds a node to the parse tree with the specified parent.

3. toString(): String:

o Returns: A string representation of the parse tree.

o Description: Generates a string representation of the parse tree, including values, fathers, left children, and right siblings. The string is suitable for debugging and understanding the structure of the parse tree.

4. subtree(node: Int): String (Private Method):

o Parameters: node - The index of the current node in the parse tree.

o Returns: A string representation of the subtree rooted at the given node.

o Description: Recursively constructs a subtree representation, including proper indentation and linking symbols.