Horvath Krisztina-Aliz & Homescu Monica Daniella

Formal-Languages-and-Compiler-Design/Lab7 Final Lab Parsing at main · krisztinahorvath/Formal-Languages-and-Compiler-Design (github.com)

Documentation

Grammar:

- Self.setOfNonterminals – set of nonterminals
- Self.setOfTerminals – set of terminals
- Self.startingSymbol – starting symbol of grammar
- Self.setOfProductions – set of productions
- Self.fileName – the name of the file from which the grammar is read

Functions:

- readGrammarFile(): reads the grammar file that is structured like: first line for the set of non-terminals, second line for the set of terminals, third line for the starting symbol and the rest are productions
- parseLine(): splits a line by spaces and returns them as a list of strings
- getSetOfNonterminals – returns the set of nonterminals
- getSetOfTerminals – returns the set of terminals
- getSetOfProductions - returns the set of productions
- getStartingSymbol - returns the starting symbol
- getProductionsOfNonterminal – returns the productions of a nonterminal
- checkCFG – checks if the grammar is a CFG by verifying if on the left side of the productions are only nonterminals.

LL(1) Parser:

- Self.firstSets – the set of firsts for all nonterminals
- Self.followSets – the set of follows for all nonterminals
- Self.grammar – grammar of the processed mini language
- Self.parseTable - the resulting parse table after parsing the given sequence
- Self.productionsNumbered – the productions numbered
- Self.alpha – input stack

- Self.beta – working stack
- Self.pi – result stack
- Self.parseTreeRoot – root of the parsing tree

Functions:
- getParseTable() – returns the parsing table
- getParseTreeRoot() – returns the parsing tree root
- setFirstSets() – goes through all the nonterminals and computes their firsts
- firstOf(nonterminal: String) – computes the first of a nonterminal(all terminals from which we can start a sequence, starting from the given nonterminal)
- setFollowSets() - goes through all the nonterminals and computes their follows
- followOf(nonterminal: String) – returns all the nonterminals into which we can go to from the given nonterminal
- createParseTable() – creates a parsing table by building a table that has as rows all nonterminals and terminals and as rows, all the terminals and $ sign in both rows and columns. Then we follow the rules given in the lectures.
- numberingProductions() – numbers all productions for all nonterminals
- pushAsChars(sequence: List<String>, stack: Stack<String>) – pushes onto the stack in reverse order all chars from the sequence
- pushIntoTree(sequence: List<String>, parent: ParseTreeNode) – adds each char from the sequence as a child of the given parent node
- initializeStacks(w: List<String>) – initializes alpha, beta, and pi
- parse(w: List<String>) – parses a given sequence 'w' using LL(1) parsing following accept and pop rules, at the same time we build the parsing tree too
- parse (w: List<String>, scanner: LexicalAnalyzer) – like the above parser but for more complex grammars, taking into consideration identifiers and constants
- processProgramInternalForm() – builds a list of strings out of the given PIF to pass it as a sequence to the parse() function
- readSequence() – reads a simple sequence line by line for a simple grammar

ParserOutput:

- Self.table – a list of ParseTreeTableRecord objects representing a parse tree table
- Self.nodeIdCounter – an integer counter to assign unique IDs to nodes in the parse tree
- Self.root – the root node of the parse tree

Functions:
- getParseTable() – returns the parsing table

- ParserOutput(root:ParseTreeNode) – constructor that initializes the parse tree table and converts the given parse tree root into the table
- converToTable() – resets the node counter, clears the table, and converts the parse tree into the table format
- traverseTree(node:ParseTreeNode, fatherId:int) – recursively traverses the parse tree, assigning IDs to nodes and constructing the table
- displayTableRecords() – displays the parse tree table records in the console
- writeToFile(filePath:String) – writes the parse tree table records to a file specified by filePath
- toStringOrRmpty(value:Object) – returns the string representation of an object or "null" if the object is null

ParseTreeTableRecord:

- Self.symbol – a string representing the symbol associated with the parse tree node
- Self.nodeId – an integer representing the unique ID of the parse tree node
- Self.fatherId – an integer representing the ID of the parent node in the parse tree
- Self.leftSiblingId – an integer representing the ID of the left sibling node in the parse tree
- Self.siblingIds – a list of integers representing the IDs of the sibling nodes in the parse tree

  Functions:
- ParseTreeTableRecord(symbol:String, nodeId:int, fatherId:int) – initializes a ParseTreeTableRecord object with the given symbol, node ID, and father ID; sets the left sibling ID to 0 and initializes the list of sibling IDs
- getSymbol() – returns the symbol of the parse tree node
- getNodeId() – returns the ID of the parse tree node
- getFatherId() – returns the ID of the parent node
- getLeftSiblingId() – returns the ID of the left sibling node
- getSiblingIds() – returns the list of sibling node IDs
- addSiblingId(siblingId:int) – adds the given sibling ID to the list of sibling IDs
- setLeftSiblingId(leftSiblingId:int) – sets the ID of the left sibling node

ParseTable:

- Self.table – a hash map storing pairs of strings as keys and pairs consisting of a list of strings and an integer as values

Functions:

- put(key: Pair<String, String>, value: Pair<List<String>, Integer>) – inserts the given key-value pair into the parse table
- get(key: Pair<String, String>) – retrieves the value associated with the given key from the parse table; returns null if the key is not found
- containsKey(key: Pair<String, String>) – checks if the parse table contains the given key; returns true if the key is found, otherwise returns false
- toString() – returns a string representation of the parse table

ParseTreeNode:

- Self.symbol – symbol of the node
- Self.parent – parent of this node
- Self.children- a list of all the children of the current node

Functions:

- getSymbol() – returns the symbol of the node
- getParents() – returns the parent of the node
- getChildren() – returns the children of the node
- addChild() – adds a child to the node
- toStringTree() – returns the current node as a string
- toStringTreeHelper() – returns in a string with more visually comprehensive symbols the relationships between the current node and its children