Rusu Horia-Radu
https://github.com/HoriaRaduRusu/Formal-Languages-And-Compiler-Design

Grammar:

Grammar is a class that describes a grammar.It holds a set of strings representing the non-terminals of the grammar, one representing the terminals, a string representing the starting symbol, and the productions are kept as a map that has as keys lists of strings, and as values lists of lists of strings - the keys represent the left sides of the productions, kept as lists where each string is one symbol, and the values represent lists of the right sides of the production - each list represents a right side, with each string being one of the symbols. It exposes the following methods:

- void loadFromFile(String file)
  Loads the grammar from the specified file. The file must have the non-terminals on the first row separated by space, the terminals on the second row separated by space, the starting symbol on the third row, and one production on each following row, with the symbols separated by space.

- String getNonTerminalSetForPrinting()
  Returns the set of non-terminals as a String to be printed.

- String getTerminalSetForPrinting()
  Returns the set of terminals as a String to be printed.

- String getProductionSetForPrinting()
  Returns the set of productions as a String to be printed.

- List<List<String>> getProductionsForNonTerminal(String nonTerminal)
  Returns a list of the right sides of the productions with the specified non-terminal as the only element in their left sides.

- String getProductionsForNonTerminaForPrintingl(String nonTerminal)
  Returns a string representing the productions with the specified non-terminal as the only element in their left sides.

- boolean checkIfCFG()
  Returns true if the grammar is a context free grammar.

- boolean isNonTerminal(String token)
  Returns true if the token represents a non-terminal in this grammar.

- int getProductionNumber(List<String> leftSide, List<String> rightSide)
  Returns the number of the production with the specified left and right sides. If the production does not appear in this grammar, throw an error.

- Map.Entry<List<String>, List<String>> getProductionWithNumber(int production)
  Returns the production with the specified number. The key of the entry is the left side, and the value is the right side. If the number is not a valid one, throw an error.

- Set<String> getNonTerminalSet()
  Returns the set of non-terminals.

- Set<String> getTerminalSet()
  Returns the set of terminals.

Item:

Item is a class describing an item used by the LR(0) parser. It keeps a string for the left side of the item, a list of strings containing the symbols before the dot, and one containing the symbols after the dot. It exposes the following methods:

- String getLeftSide()
  Returns the left side of the item.

- List<String> getBeforeDot()
  Returns the list of symbols before the dot.

- String getFirstAfterDot()
  Returns the first symbol after the dot, or an empty string if there is no symbol after the dot.

- Item shiftLeft()
  Returns a new item, representing the form of the item if the first symbol after the dot is moved before the dot. If there is no symbol after the dot, return the same item.

- String toString()
  Returns the string representation of the item.

Parser:

Parser is a class describing a LR(0) parser. It has the following methods:

- Set<Item> closure(Set<Item> argument)
  Computes the closure of the item set given as argument.

- Set<Item> goTo(Set<Item> state, String symbol)
  Computes the goTo function for the given state and symbol, and returns the resulting state as a set of items.

- Map<String, Set<Item>> getCanonicalCollection()
  Computes the canonical collection of the given grammar. The keys of the maps represent the notations of the states, and the values represent the actual states.

- Map<String, Map<String, String>> createTable()
  Creates the parsing table for the given grammar. Each map entry represents a row of the table, with the key being the state it represents. Each value map has as keys “ACTION”, which has as value the action for the respective state, and every symbol

in the grammar, which have as values the goTo state for the given state and symbol, if one exists.

- ParserOutput parse(List<String> sequence)
  Parses the given sequence (which has every symbol as a different string), and returns an instance of the ParserOutput class.

- void shift(Stack<String> workingStack, Stack<String> inputStack, Map<String, Map<String, String>> table)
  Does the shift operation with the given parameters. Removes the top element on the input stack, adds it to the working stack, and also adds the state in which we shift to the top of the working stack.

- void reduce(Stack<String> workingStack, Stack<String> outputStack, int productionNumber, Map<String, Map<String, String>> table)
  Does the reduce operation with the given parameters. Gets the production with the given number, removes the top-most 2*p elements from the working stack (where p is the number of elements in the right side of the production), adds the left side of the production to the top of the working stack, adds the state created by the goTo function with the top-most state and symbol from the working stack to the same stack, and adds the production number to the output stack.

- String determineAction(Item item)
  Returns the string representing the action described by the item - accept, reduce, shift or error.

- String determineGoTo(Set<Item> startingState, String startingToken, Map<String, Set<Item>> states)
  Returns the string representing the state which is the result of the goTo function applied on the given state and token.

Node:

A data class keeping a node in the parsing tree. It keeps a string for the symbol of the node, a Node as the parent of the node in the tree, and a list of nodes as the children of the node, in order from the left-most to the right-most. It has the following methods:

- String getSymbol()
  Returns the symbol of the node.

- Node getParent()
  Returns the parent of the node.

- List<Node> getChildren()
  Returns the children of the node.

- void addChild(Node node)
  Adds a node to the list of children of the node.

TableConstructData:

A data class keeping the information necessary when converting the parse tree to the table representation. It keeps a Node, describing the node whose information needs to be added to the table, an integer representing the index of the parent of the node in the table, and a boolean which is true if the node has a right sibling. It has the following methods:

- Node getNode()
  Returns the node.

- Integer getParentIndex()
  Returns the index of the parent of the node in the table.

- boolean hasRightSibling()
  Returns true if the element has a right sibling.

TableRow:

A data class keeping the information kept in a row of the table representation of the parser output. It keeps a string, describing information from the table, an integer representing the index of the parent, and an integer representing the index of the right sibling. Either integer will have the value -1 if the described node has no parent, respectively a right sibling. It has the following methods:

- String getInfo()
  Returns the information of the row.

- Integer getParentIndex()
  Returns the index of the parent of the node in the table.

- Integer getParentIndex()
  Returns the index of the right sibling of the node in the table.

- String toString()
  Returns the string representation of the table row.

TableOutput:

A data class keeping the information kept in the table representation of the parser output. It keeps a list of TableRows, such that the index of an item in the list represents the index of the item in the table. It has the following method:

- String toString()
  Returns the string representation of the table.

ParserOutput:

A class representing the output of the parsing of a sequence. It has the grammar with which the parsing was done, a stack representing the string of productions, a string representing whether the parsing result is “success” or “error”, and a TableOutput which keeps the table representation of the output if the result is “success”. It has the following methods:

- void printTableToConsole()
  Prints the result and the output table to the console.

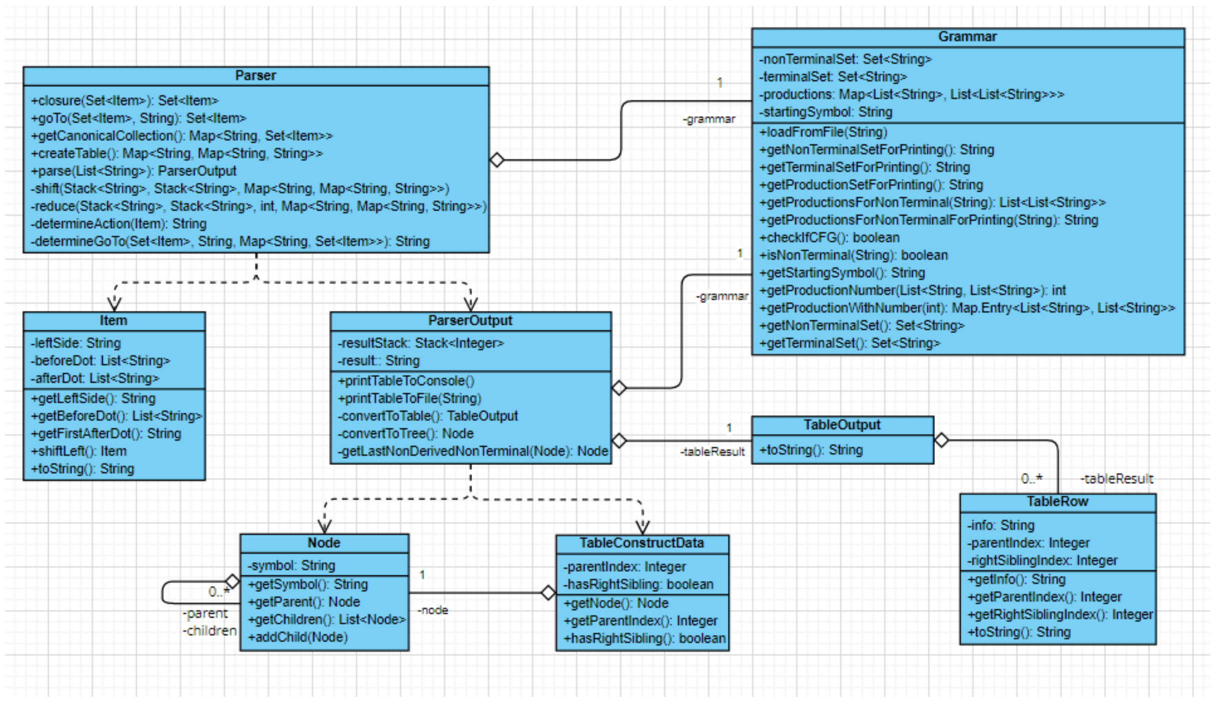
- void printTableToFile(String fileName)
  Prints the result and the output table to a file with the specified name.

- TableOutput convertToTable()
  Converts the string of productions to the table representation.

- Node convertToTree()
  Converts the string of productions to the parsing tree, and returns the root node of the tree.

- Node getLastNonDerivedNonTerminal(Node rootNode)
  Returns the Node representing the right-most non-derived non-terminal in the tree with the given root.

Class diagram:


Parser
+closure(Set<Item>): Set<Item>
+goTo(Set<Item>, String): Set<Item>
+getCanonicalCollection(): Map<String, Set<Item>>
+createTable(): Map<String, Map<String, String>>
+parse(List<String>): ParserOutput
-shift(Stack<String>, Stack<String>, Map<String, Map<String, String>>)
-reduce(Stack<String>, Stack<String>, int, Map<String, Map<String, String>>)
-determineAction(Item): String
-determineGoTo(Set<Item>, String, Map<String, Set<Item>>): String
Grammar
-nonTerminalSet: Set<String>
-terminalSet: Set<String>
-productions: Map<List<String>, List<List<String>>>
-startingSymbol: String
+loadFromFile(String)
+getNonTerminalSetForPrinting(): String
+getTerminalSetForPrinting(): String
+getProductionSetForPrinting(): String
+getProductionsForNonTerminal(String): List<List<String>>
+getProductionsForNonTerminalForPrinting(String): String
+checkIfCFG(): boolean
+isNonTerminal(String): boolean
+getStartingSymbol(): String
+getProductionNumber(List<String, List<String>>): int
+getProductionWithNumber(int): Map.Entry<List<String>, List<String>>
+getNonTerminalSet(): Set<String>
+getTerminalSet(): Set<String>
1
-grammar
1
-grammar
Item
-leftSide: String
-beforeDot: List<String>
-afterDot: List<String>
+getLeftSide(): String
+getBeforeDot(): List<String>
+getFirstAfterDot(): String
+shiftLeft(): Item
+toString(): String
ParserOutput
-resultStack: Stack<Integer>
-result:: String
+printTableToConsole()
+printTableToFile(String)
-convertToTable(): TableOutput
-convertToTree(): Node
-getLastNonDerivedNonTerminal(Node): Node
TableOutput
+toString(): String
1
-tableResult
0..*
-tableResult
TableRow
-info: String
-parentIndex: Integer
-rightSiblingIndex: Integer
+getInfo(): String
+getParentIndex(): Integer
+getRightSiblingIndex(): Integer
+toString(): String
Node
-symbol: String
+getSymbol(): String
+getParent(): Node
+getChildren(): List<Node>
+addChild(Node)
0..*
-parent
-children
1
-node
TableConstructData
-parentIndex: Integer
-hasRightSibling: boolean
+getNode(): Node
+getParentIndex(): Integer
+hasRightSibling(): boolean