**Documentation for LR(0) Parsing Algorithm and Tree Generation**

**Overview**

This document provides detailed documentation for the LR class, which implements the LR(0) parsing algorithm, and the Tree class, which handles the generation of a parse tree. These classes are designed to work together to parse input strings based on a given grammar and produce a corresponding parse tree.

**LR Class**

The LR class implements the LR(0) parsing algorithm. It takes a grammar as input and performs parsing by computing closure and goto functions, generating the canonical collection of LR(0) items, and parsing an input string based on the LR table.

**Constructor**

public LR(Grammar grammar)

* **Description**: Initializes the LR parser with the provided grammar. It creates an enriched grammar and retrieves the ordered productions.
* **Parameters**:
  + grammar: The grammar object used for parsing.

**Methods**

public String getNonTerminalBeforeDot(Item item)

* **Description**: Retrieves the non-terminal immediately before the dot in the provided item.
* **Parameters**:
  + item: The Item object to be checked.
* **Returns**: The non-terminal before the dot, or null if it is not a non-terminal.

public State closure(Item item)

* **Description**: Computes the closure of a given item by adding items to the state until no more items can be added.
* **Parameters**:
  + item: The Item object for which the closure is computed.
* **Returns**: A State object representing the closure.

public State goTo(State state, String elem)

* **Description**: Computes the goto function for a given state and symbol. The goto function is used to transition between states based on the next symbol in the input.
* **Parameters**:
  + state: The current State object.
  + elem: The symbol to move to.
* **Returns**: A new State object resulting from the goto function.

public CanonicalCollection getCanonicalCollection()

* **Description**: Generates the canonical collection of LR(0) items. This collection includes all possible states that can be reached in the parsing process.
* **Returns**: A CanonicalCollection object containing all states and transitions.

public void parse(Stack<String> inputStack, LrTable lrTable, String filePath) throws IOException

* **Description**: Parses the input stack using the LR(0) table and writes the output to a file. The parsing algorithm operates by shifting symbols onto the stack and reducing them based on the LR(0) table, following the LR(0) parsing approach.
* **Parameters**:
  + inputStack: A Stack containing the input symbols.
  + lrTable: The LR(0) table used for parsing.
  + filePath: The path to the file where output will be written.
* **Detailed Steps**:
  + **Shift**: Moves the current input symbol to the working stack and transitions to the next state.
  + **Reduce**: Replaces a sequence of symbols on the working stack with a non-terminal, based on a production rule, and transitions to the corresponding state.
  + **Accept**: Confirms that the input string is accepted by the grammar.

**Tree Class**

The Tree class generates a parse tree from a list of production indices derived from the parsing process.

**Constructor**

public Tree(Grammar grammar)

* **Description**: Initializes the Tree with the provided grammar.
* **Parameters**:
  + grammar: The grammar object used to guide tree generation.

**Methods**

public TreeRow generateTree(List<Integer> input)

* **Description**: Generates the root of the parse tree from a list of production indices. This method starts the tree construction by using the first production in the list and recursively builds the tree for each symbol.
* **Parameters**:
  + input: A List of integers representing the indices of productions used in parsing.
* **Returns**: The root TreeRow of the generated tree.

public TreeRow buildFromParent(int level, TreeRow parent, List<String> rhs, List<Integer> input)

* **Description**: Recursively builds the parse tree from the parent node. This method processes each symbol in the right-hand side (RHS) of a production, creating child nodes for non-terminals and siblings for terminals.
* **Parameters**:
  + level: The current level in the tree.
  + parent: The parent TreeRow node.
  + rhs: The right-hand side of the production.
  + input: A List of integers representing production indices.
* **Returns**: The constructed TreeRow node.

public void printTree(TreeRow node, String filePath) throws IOException

* **Description**: Prints the tree structure to the console and writes it to a file. The method traverses the tree level by level, capturing each node's details and outputting them in a readable format.
* **Parameters**:
  + node: The root TreeRow of the tree.
  + filePath: The path to the file where the tree structure will be written.