<https://github.com/cs-ubbcluj-ro/lab-work-computer-science-2024-dragosgavrus1/tree/main/3-Parser>

Gavrus Dragos Andrei

**Parser Class Documentation**

**The Parser class implements an LR(0) parsing algorithm, generating a canonical collection of LR(0) items and building a parsing table to process input sequences according to a given grammar. Below is a comprehensive description of the class and its functionality:**

**Attributes**

* **grammar: The grammar to be parsed, containing non-terminals, terminals, a start symbol, and productions.**
* **items: A list of all items (productions with dot positions) derived from the grammar.**
* **canonicalCollection: A collection of unique LR(0) states representing all possible configurations during parsing.**
* **table: A parsing table used to store the shift, reduce, and accept actions, as well as transitions between states.**

**Methods**

**\_\_init\_\_(self, grammar)**

* **Initializes the parser with the given grammar.**
* **Augments the grammar by adding a new start production and computes the initial set of LR(0) items.**

**addAugmentedProduction(self)**

* **Adds an augmented production S' → S to the grammar, where S is the original start symbol.**
* **Ensures the parsing process begins with a unique start rule.**

**computeInitialLr0Items(self)**

* **Creates an initial list of LR(0) items for all productions in the grammar.**
* **Each item has a dot placed at the start of the production.**

**closure(self, itemList)**

* **Computes the closure of a given set of LR(0) items:**
  + **Adds items for all productions of non-terminals that appear immediately after the dot.**
  + **Stops when no new items can be added.**
* **Returns a new State object representing the closure.**

**isInCanonicalCollection(self, state)**

* **Checks if a given state is already part of the canonical collection.**
* **Returns True if the state exists, otherwise False.**

**computeCanonicalCollection(self)**

* **Constructs the canonical collection of LR(0) states:**
  1. **Starts with the closure of the augmented production.**
  2. **Iteratively computes transitions (goTo) for all symbols and adds new states to the collection.**
  3. **Stops when no new states are added.**
* **Prints the canonical collection.**

**goTo(self, state, symbol)**

* **Computes the set of items obtained by shifting the dot over a given symbol in the provided state.**
* **Returns the closure of the resulting set of items or an empty list if no transition is possible.**
* **Updates the parsing table with state transitions.**

**computeTableActions(self)**

* **Populates the parsing table with actions (shift, reduce, accept) for each state:**
  + **shift: When a transition exists for a terminal symbol.**
  + **reduce: When a production is completed (dot at the end).**
  + **accept: When the augmented production is completed.**

**buildInputStack(self, sequence)**

* **Builds a stack of symbols from the input sequence by identifying terminal and non-terminal symbols in order.**

**getStateHavingIndex(self, index)**

* **Retrieves a state from the canonical collection based on its index.**

**parseSequence(self, sequence)**

* **Parses a given sequence using the LR(0) parsing table:**
  1. **Maintains a working stack, input stack, and output stack.**
  2. **Iteratively processes actions from the parsing table:**
     + **shift: Adds the symbol and its target state to the working stack.**
     + **reduce: Replaces symbols on the stack with the left-hand side of a production and transitions to a new state.**
     + **accept: Terminates the parsing process if the input is valid.**
  3. **Throws an error if the sequence cannot be parsed.**
* **Prints the working stack, input stack, and output stack during parsing.**

**printCanonicalCollection(self)**

* **Prints the canonical collection of states in a readable format.**