https://github.com/cs-ubbcluj-ro/lab-work-computer-science-2024-carla-mirea/tree/main/3-Parser/Lab7_ParserPart3/Lab7_ParserPart3

The `Grammar` class provides functionality to represent and manipulate context-free grammars (CFGs). It supports reading grammar definitions from a file, checking CFG validity, and querying grammar elements such as productions and symbols.

`__init__(self)`

- Initializes the grammar with the following attributes:
    - `non_terminals`: A list of non-terminal symbols.
    - `terminals`: A list of terminal symbols.
    - `start_symbol`: The starting symbol of the grammar.
    - `productions`: A dictionary mapping non-terminals to their production rules.

`load_from_file(self, file_name: str)`

- Reads a grammar definition from a file and populates the grammar's attributes.

`check_if_CFG(self) -> bool`

- Validates if the grammar adheres to the rules of a context-free grammar (CFG):
    - All left-hand sides (LHS) are single non-terminals.
    - All symbols in the right-hand side (RHS) are in the grammar's alphabet.
    - The start symbol is defined in the grammar.
- Returns:
    - `True` if valid CFG.
    - `False` otherwise.

`get_productions_for_non_terminal(self)`

- Prompts the user for a non-terminal and prints its productions.
- Checks if the input is a valid non-terminal and provides an error message if invalid.

`__str__(self)`

- Provides a string representation of the grammar, listing non-terminals, terminals, the start symbol, and production rules.

Class: ACTION (Enum)

An enumeration that defines different types of actions that can occur during the parsing process:

- SHIFT (1): Indicates a "shift" action, where a symbol is pushed onto the stack.
- ACCEPT (2): Indicates that the parser has successfully accepted the input string.
- REDUCE (3): Indicates a "reduce" action, where a production rule is applied to reduce the stack.
- REDUCE_REDUCE_CONFLICT (4): Indicates a conflict between two possible reductions.
- SHIFT_REDUCE_CONFLICT (5): Indicates a conflict between a shift and a reduce action.

`Class: State`

Represents a state in the parser's state machine, which stores information about the closure (set of items), the action associated with the state, and methods for determining the action.

Methods:

- `__init__(self, closure_items, closure, enrichedSymbol)`: Initializes the state with the closure items, the closure itself, and sets the action based on the closure and enriched symbol.
- `set_action(self, enrichedSymbol)`: Determines the action for the state based on the closure:
  - ACCEPT if the closure represents a complete production.
  - REDUCE if the dot has reached the end of the production.
  - SHIFT if the dot is not at the end and there are symbols to shift.
  - REDUCE_REDUCE_CONFLICT or SHIFT_REDUCE_CONFLICT if conflicts exist.
- `check_all_not_dot_end(self) -> bool`: Checks whether all items in the closure are not at the end of their productions (i.e., whether the dot is not at the end).
- `check_all_dot_end(self) -> bool`: Checks whether all items in the closure have their dot at the end of their production.
- `get_all_symbols_after_dot(self)`: Returns a list of symbols that appear immediately after the dot in the closure items.
- `__eq__(self, other)`: Compares two states for equality based on their closure items.
- `__str__(self)`: Returns a string representation of the state, displaying its ID, closure items, and the closure itself.

Class: `Connection`

Represents a connection (or transition) between two states based on a symbol in the parsing process.

Methods:

- `__str__(self)`: Returns a string representation of the connection, displaying the starting state, final state, and symbol.

Class: `ProductionItem`

Represents a single item in a production rule, with a dot indicating the current position during parsing.

`Methods:`

- `__init__(self, lhs: str, rhs: list, dot_position: int)`: Initializes the ProductionItem with the left-hand side, right-hand side, and the dot position.
- `__eq__(self, other)`: Compares two ProductionItem objects for equality, checking if their lhs, rhs, and dot_position are the same.
- `__str__(self)`: Returns a string representation of the production item, showing the production rule with a dot indicating the current position.

Class: `Parser`

- `Methods:`
  - `closure()`: Computes the closure of a set of items by iterating over them and adding new items derived from the grammar rules.
  - `goto()`: Computes the next state by applying a grammar symbol to the current state.
  - `create_canonical_collection()`: Builds the entire set of possible states for the parser by repeatedly applying closure and goto operations.
  - `create_parsing_table()`: Populates the parsing table with actions (shift, reduce, accept) for each state.
  - `parse_sequence()`: Takes an input sequence and parses it using the created parsing table, returning a list of production IDs.

Class: `ParserOutput`:

- **`compute_parsing_tree()`**: Constructs a parsing tree based on the output band (sequence of production IDs).
- **`__check_has_children()`**: Checks if a node already has children.
- **`print_to_file()`**: Writes the parsing tree to a file.