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

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.

Parser Class

The `Parser` class provides functionality to perform LR(0) parsing operations on a given context-free grammar (CFG). It supports generating canonical collections of LR(0) items, computing closures, and determining transitions for symbols.

`__init__(self, grammar: Grammar)`

Initializes the parser with the following attributes:

- **`grammar`**: An instance of the `Grammar` class that represents the CFG to be parsed.
- **`canonical_collection`**: A list of `State` objects, representing the canonical collection of LR(0) items.

`closure(self, items: list) -> State`

Computes the closure of a given set of LR(0) items.
Parameters:

- **`items`**: A list of `ProductionItem` objects representing the initial items for closure computation.

Returns:

- A `State` object containing the closure of the given items.

`goto(self, state: State, symbol: str) -> State`

Computes the transition (goto) for a given state and symbol.
Parameters:

- **`state`**: A `State` object representing the current state.
- **`symbol`**: A string representing the grammar symbol (terminal or non-terminal).

Returns:

- A `State` object representing the resulting state after transitioning on the given symbol.

`create_canonical_collection(self)`

Generates the canonical collection of LR(0) items for the grammar.
Iterates over states in the canonical collection, computing closures and transitions for all symbols until no new states are generated.

`is_item_in_closure(item, closure) -> bool`

Checks if a `ProductionItem` is already present in a given closure.
Parameters:

- **`item`**: A `ProductionItem` object to check.
- **`closure`**: A list of `ProductionItem` objects representing the closure.

Returns:

- `True` if the item is already in the closure.
- `False` otherwise.

Nested Classes

`ProductionItem`

Represents a production rule item with a dot position for LR(0) parsing.
Attributes:

- **`lhs`**: A string representing the left-hand side of the production.
- **`rhs`**: A list of strings representing the right-hand side of the production.
- **`dot_position`**: An integer representing the position of the dot in the production.

Methods:

- **`__eq__(self, other)`**: Compares two `ProductionItem` objects for equality.
- **`__str__(self)`**: Provides a string representation of the production with the dot position.

`State`

Represents a state in the canonical collection, containing closure items and transitions.
Attributes:

- **`id`**: A unique integer ID for the state.
- **`closure_items`**: A list of `ProductionItem` objects representing the initial items of the state.
- **`closure`**: A list of `ProductionItem` objects representing the computed closure of the state.

Methods:

- **`get_symbols_after_dot(self) -> list`**: Retrieves a list of symbols that appear immediately after the dot in the closure items.
- **`__eq__(self, other)`**: Compares two `State` objects for equality.
- **`__str__(self)`**: Provides a string representation of the state, including its ID, closure items, and transitions.