File: `Grammar.py`

Purpose

The `Grammar.py` file handles the representation, validation, and processing of context-free grammars. It supports reading grammars from files and provides utility methods for working with grammar components.

Class: `Grammar`

Attributes:

- `N` (set): The set of non-terminal symbols.
- `E` (set): The set of terminal symbols.
- `S` (str): The start symbol.
- `P` (dict): Productions, where keys are non-terminals and values are lists of lists (alternatives).

Methods:

1. **Grammar Construction**:
    - **`from_file(filename)`**: Reads a grammar file and constructs a `Grammar` object.

Format:
css
Copy code

```
N = S A B
E = a b c
S = S
P =
S -> A b | c
A -> a
```

        - ■
    - **`parse_line(line)`**: Parses a single line to extract non-terminals or terminals.
    - **`parse_productions(lines)`**: Parses production rules into a dictionary.
2. **Validation**:
    - **`validate(N, E, S, P)`**: Checks whether the grammar is a valid context-free grammar.
        - Ensures the start symbol is in NNN.
        - Ensures production rules have valid symbols.
    - **`is_cfg()`**: Determines if the grammar is a context-free grammar.

3. **Utility Methods**:
    - `get_nonterminal_productions(nonterminal)`: Retrieves the productions for a specific non-terminal.
    - `get_nonterminals()` and `get_terminals()`: Return the sets of non-terminals and terminals, respectively.
4. **String Representation**:
    - `__str__()`: Returns a formatted string representation of the grammar.
5. **Interactive Grammar Processing**:
    - `process_grammar(filename)`: Reads and processes a grammar file, printing its details and validation status.
    - `process_grammar_with_interface(filename)`: Provides an interactive interface for exploring grammar properties.

File: `Parser.py`

Purpose

The `Parser.py` file implements an LR(0) parser for context-free grammars. It builds the LR(0) automaton, generates parsing tables, and parses input strings based on the grammar's productions.

Classes

1. LR0Item

Represents an individual item in the LR(0) parsing automaton.

- **Attributes**:
    - `lhs` (str): The left-hand side of the production.
    - `rhs` (list): The right-hand side of the production as a list of symbols.
    - `dot_position` (int): The position of the dot in the item, indicating progress.
- **Methods**:
    - `__eq__` and `__hash__`: Enable comparison and use in sets.
    - `__repr__`: Returns a string representation (e.g., `A -> a . B c`).

2. LR0State

Represents a state in the LR(0) automaton.

- **Attributes**:
    - `items` (set): A set of `LR0Item` objects that make up the state.
- **Methods**:
    - `__eq__` and `__hash__`: Enable comparison and use in dictionaries.

    - `__repr__`: Returns a string representation of the state.

3. LR0Parser

Implements the logic for building the LR(0) automaton and parsing tables.

- **Attributes**:
    - `grammar` (Grammar): The grammar object used for parsing.
    - `states` (list): The list of LR(0) states.
    - `transitions` (dict): Transitions between states based on symbols.
    - `action_table` (dict): The action table for parsing (shift, reduce, accept).
    - `goto_table` (dict): The GOTO table for non-terminal transitions.
- **Methods**:
    - **`closure(items)`**: Computes the closure of a set of items.
    - **`goto(items, symbol)`**: Computes the GOTO of a set of items on a given symbol.
    - **`canonical_collection()`**: Constructs the canonical collection of LR(0) states.
    - **`build_automaton()`**: Builds the LR(0) automaton.
    - **`build_parsing_table()`**: Constructs the action and GOTO tables.
    - **`parse(input_string)`**: Parses an input string and determines if it's accepted.

Workflow

1. **Automaton Construction**:
    - The canonical collection of LR(0) states is built using the `closure` and `goto` methods.
    - Transitions between states are computed and stored.
2. **Parsing Table Construction**:
    - The `action_table` specifies whether to shift, reduce, or accept based on states and input symbols.
    - The `goto_table` specifies transitions for non-terminals.
3. **Input Parsing**:
    - Simulates the LR(0) parsing process using a stack and the parsing tables.

File Interactions

1. **Grammar File Input (`g1.txt`)**:
    - The `Grammar` class reads the grammar from the file.
    - It validates and parses the grammar into its components.
2. **Parsing Process**:
    - The `LR0Parser` builds the automaton and parsing tables using the grammar.
    - It parses an input string (e.g., `seq.txt`) to determine if it belongs to the grammar's language.

3. **Output**:
    - The parsing result and the tables are written to `out1.txt`.

Example Workflow

Grammar File (`g1.txt`):
css
Copy code

```
N = S A
E = a b c
S = S
P =
S -> a A
A -> b A | c
```

1.

Input String (`seq.txt`):
Copy code

```
abbcc
```

2.
3. **Execution**:
    - The `Grammar` object is constructed and validated.
    - The `LR0Parser` builds the automaton and parsing tables.
    - The input string is parsed, and the result is written to `out1.txt`.

Output (`out1.txt`):
css
Copy code

```
Input string is accepted.
+-------+-------+-------+-------+-------+
| State |   a   |   b   |   c   |   $   |
+-------+-------+-------+-------+-------+
|   0   | shift |       |       |       |
...
```

4.