Documentation

Implement the LL(1) parsing algorithm implementing FIRST and FOLLOW functions.

```
+-----------------------+
|    LL1Parser          |
+-----------------------+
| - follow_sets         |
| - grammar             |
| - parse_table         |
+-----------------------+
| + __init__            |
| + build_parse_table   |
| + find_productions    |
| + find_rule_for_follow|
| + find_rule_for_firsts|
| + first               |
| + first_if_epsilon    |
| + follow              |
| + parse_sequence      |
+-----------------------+
```

Method: build_parse_table(self)

Parameters: None

Functionality:

- Builds the LL(1) parse table based on the provided grammar.
- Calls the **follow** method to fill up the follow sets for each nonterminal.
- Calls the **find_production** method to find the corresponding rule and index of a given <terminal, nonterminal> pair.

Method: find_production(self, nonterminal, terminal)

Parameters:

- **nonterminal**: The nonterminal symbol.
- **terminal**: The terminal symbol.

Returns:

- A tuple representing the production rule and its number if a production is found; otherwise, returns **None**.

Functionality:

- Finds the production rule for a given nonterminal and terminal.
- For every <terminal, terminal> pair returns "pop" or except if terminal is "$"
- Checks for first(nonterminal) and returns the rule corresponding to the <nonterminal, terminal> pair, otherwise it looks into follow(nonterminal)

Method: find_rule_for_firsts(self, nonterminal, terminal)

Parameters:

- **nonterminal**: The nonterminal symbol.
- **terminal**: The terminal symbol.

Returns:

- A tuple representing the production rule and its number if a production is found; otherwise, returns **None**.

Functionality:

- Finds the production rule for a given <nonterminal, terminal> looking in the first table

Method: find_rule_for_follows(self, nonterminal, terminal)

Parameters:

- **nonterminal**: The nonterminal symbol.
- **terminal**: The terminal symbol.

Returns:

- A tuple representing the production rule and its number if a production is found; otherwise, returns **None**.

Functionality:

- Similarly to **find_rule_for_firsts** it looks for a given terminal in the follow table for nonterminal

Method: first(self, symbol, from_term)

Parameters:

- **symbol**: The symbol for which to find the first set.
- **from_term**: The production term from which the **symbol** originates.

Returns:

- A set representing the first set of the given symbol.

Functionality:

- Computes first of a given symbol, considering epsilon productions recursively. If terminal the recursion ends, otherwise it keeps looking.

Method: first_if_epsilon(self, term, symbol)

Parameters:

- **term**: The production term.
- **symbol**: The symbol for which to find the first set if epsilon.

Returns:

- The next symbol in the production term after the occurrence of the given symbol.

Functionality:

- Finds the next symbol in the production term after the occurrence of the given symbol. If nothing is found None is returned.

Method: follow(self, nonterminal)

Parameters:

- **nonterminal**: The nonterminal symbol.

Returns:

- A set representing the follow set of the given nonterminal.

Functionality:

- Computes the follow set of a given nonterminal using the follow sets.

Method: parse_sequence(self, sequence)

Parameters:

- **sequence**: The input sequence to be parsed.

Returns: None

Functionality:

- Parses the input sequence using the LL(1) parsing algorithm.
- While the input stack and working stack are not both "$" it keeps looking for rules to reduce the input stack to "$". If a rule is found for the first symbol of the work stack it adds it to the work stack and when first symbols match in the input stack and symbol stack they get popped from both stacks.

Example

Start: S

Non-terminals: S A B C D

Terminals: a + * ( )

Production:

S ::= B A

A ::= + B A

A ::= epsilon

B ::= D C

C ::= * D C | epsilon

D ::= (S) | a

Sequence: "a * (a + a) $"

Result output: 1485714862486363