https://github.com/trutadan/University-Work/tree/main/Semester%205/Formal%20Languages%20and%20Compiler%20Design/lab06

Recursive Descendant Parsing Implementation

Recursive Descendant Parsing is a top-down parsing technique that starts with the topmost grammar rule and recursively explores the production rules to match the input string(sequence).

Class Structure

Attributes

Grammar (__grammar): Represents the grammar to be parsed, provided as an instance of the Grammar class.

Current State (__current_state): Tracks the current state of the parsing process: 'q' for normal, 'b' for backtracking, 'f' for final success, and 'e' for error.

Index Position (__index_position): Indicates the current position in the input string(sequence).

Working Stack (__working_stack): Maintains a stack of symbols being processed during parsing.

Input Stack (__input_stack): Represents the input string as a stack of symbols.

Methods

expand()

Expands the current state by moving the input stack's first element (non-terminal) to the working stack, along with its first production to the input stack.

advance()

Advances the current state by moving the input stack's first element (terminal) to the working stack and increments the index position.

momentary_insuccess()

Modifies the current parsing state to the back state ('b').

back()

Backtracks the current state by popping the working stack's last element and moving it back to the input stack.

another_try()

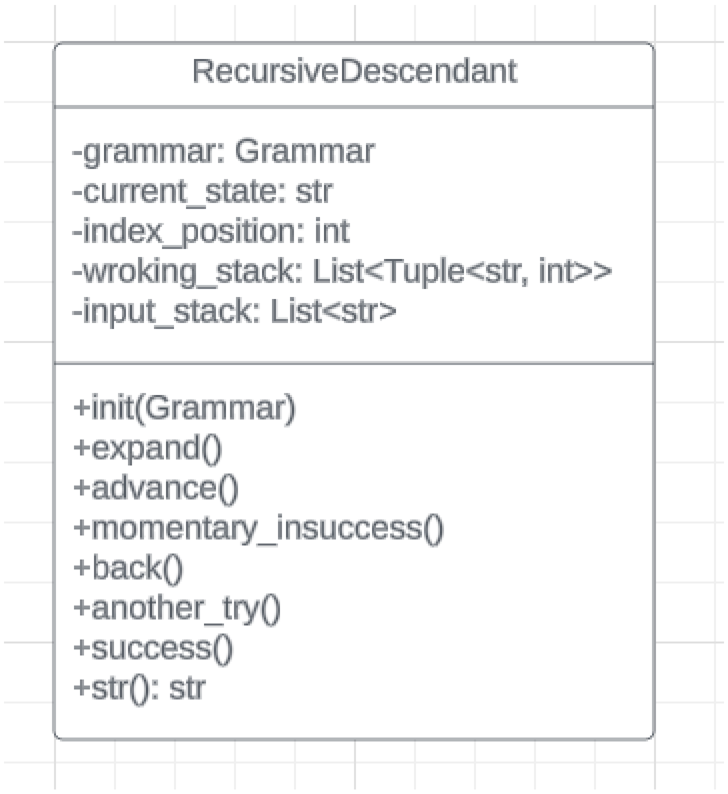
Modifies the current parsing state based on the non-terminal's productions:

1) If more productions exist, the state is set to normal, and the next production is moved to the input stack.
2) If no more productions exist:
    - If the non-terminal is the start symbol and the index position is 1, the state is set to error.
    - Otherwise, the non-terminal is moved back to the input stack, and the state remains in the back state.

success()

Sets the current state to final success ('f').

The class diagram:


RecursiveDescendant
-grammar: Grammar
-current_state: str
-index_position: int
-wroking_stack: List<Tuple<str, int>>
-input_stack: List<str>
+init(Grammar)
+expand()
+advance()
+momentary_insuccess()
+back()
+another_try()
+success()
+str(): str