Logic for FIRST and FOLLOW.

1. Define the grammar rules. The grammar is represented as a dictionary, where each key represents a non-terminal symbol and each value is a list of its production rules.

```
grammar = {
 "S": [["A", "B", "C"]],
 "A": [["a", "b", "A'"], ["a", "b"]],
 "A'": [["a", "b", "A'"], []],
 "B": [["b", "B'"], ["b"]],
 "B'": [["C", "B'"], []],
 "C": [["c"], ["c", "C"]]
}
```

2. Initialize the FIRST and FOLLOW sets for each non-terminal symbol. Two dictionaries are created to store the FIRST and FOLLOW sets for each non-terminal symbol. The sets are initially empty.

```
first_sets = {}
follow_sets = {}
for symbol in grammar.keys():
 first_sets[symbol] = set()
 follow_sets[symbol] = set()
```

```

3. Define a function to compute the FIRST set of a symbol. This function takes a non-terminal symbol as input and returns its FIRST set. If the FIRST set for the symbol has already been computed, the function simply returns it. Otherwise, the function computes the FIRST set for each production rule of the symbol, following these steps:

- If the production rule starts with a non-terminal symbol, compute its FIRST set recursively. If the non-terminal symbol can't derive epsilon, stop processing the production.
- If all symbols in the production rule can derive epsilon, add epsilon to the FIRST set.
- After all production rules have been processed, the function returns the FIRST set for the symbol.

4. Define a function to compute the FOLLOW set of a symbol. This function takes a non-terminal symbol as input and computes its FOLLOW set. If the symbol is the start symbol, it adds the end-of-string marker ($) to its FOLLOW set. Then, for each production rule that contains the symbol, the function follows these steps:

- If the symbol is the last symbol in the production rule, it adds the FOLLOW set of the head symbol to its FOLLOW set (unless the head symbol is the same as the current symbol, in which case we avoid infinite recursion).
- Otherwise, it adds the FIRST set of the symbol's successor to its FOLLOW set. If the successor is a non-terminal symbol that can derive epsilon, it also adds the FOLLOW set of the head symbol to its FOLLOW set.

5. Compute the FIRST and FOLLOW sets for each non-terminal symbol:

6. Print the FIRST and FOLLOW sets for each non-terminal symbol. Finally, the FIRST and FOLLOW sets for each non-terminal symbol are printed to the console, using the join function to format the sets as comma-separated strings. The sets are sorted for easier readability.

```
grammar = {
    "S": [["A", "B", "C"]],
    "A": [["a", "b", "A'"], ["a", "b"]],
    "A'": [["a", "b", "A'"], []],
    "B": [["b", "B'"], ["b"]],
    "B'": [["C", "B'"], []],
    "C": [["c"], ["c", "C"]]
}
first_sets = {}
follow_sets = {}
for symbol in grammar.keys():
    first_sets[symbol] = set()
    follow_sets[symbol] = set()
# Function to compute the FIRST set of a symbol
def compute_first(symbol):
    if first_sets[symbol]:
        return first_sets[symbol]
    for production in grammar[symbol]:
        for i in range(len(production)):
            if production[i] not in grammar.keys():
                first_sets[symbol].add(production[i])
```
```

```
                break
            else:
                first_sets[symbol].update(compute_first(production[i]))
                if "epsilon" not in first_sets[production[i]]:
                    break
        else:
            first_sets[symbol].add("epsilon")
    return first_sets[symbol]
# Function to compute the FOLLOW set of a symbol
def compute_follow(symbol):
    if symbol == "S":
        follow_sets[symbol].add("$")
    for head, productions in grammar.items():
        for production in productions:
            try:
                symbol_index = production.index(symbol)
            except ValueError:
                continue
            if symbol_index == len(production) - 1:
                if head != symbol:
                    follow_sets[symbol].update(compute_follow(head))
            else:
                next_symbol = production[symbol_index + 1]
                follow_sets[symbol].update(compute_first(next_symbol) - set("epsilon"))
                if "epsilon" in compute_first(next_symbol):
                    follow_sets[symbol].update(compute_follow(head))
    follow_sets[symbol].discard("epsilon") # Remove epsilon from FOLLOW sets
    return follow_sets[symbol]
# Call the functions to compute the FIRST and FOLLOW sets for each symbol in the grammar
for symbol in grammar.keys():
    compute_first(symbol)
    compute_follow(symbol)
# Print the FIRST and FOLLOW sets for each non-terminal symbol
for symbol in grammar.keys():
    print(f"FIRST({symbol}) = {first_sets[symbol]}")
    print(f"FOLLOW({symbol}) = {follow_sets[symbol]}")
```

```
FIRST(S) = {'a'}
FOLLOW(S) = {'$'}
FIRST(A) = {'a'}
FOLLOW(A) = {'b'}
FIRST(A') = {'epsilon', 'a'}
FOLLOW(A') = {'b'}
FIRST(B) = {'b'}
FOLLOW(B) = {'c'}
FIRST(B') = {'c', 'epsilon'}
FOLLOW(B') = {'c'}
FIRST(C) = {'c'}
FOLLOW(C) = {'$', 'c'}
```

```
# Define the parsing table as a dictionary of dictionaries
parsing_table = {}
# Populate the parsing table
parsing_table[('S', 'a')] = ['A', 'B']
parsing_table[('A', 'a')] = ['a', "A'"]
parsing_table[('A', 'b')] = ['epsilon']
parsing_table[("A'", 'epsilon')] = ['epsilon']
parsing_table[("A'", 'a')] = ['a', "A'"]
parsing_table[('B', 'b')] = ['b', "B'"]
parsing_table[("B'", 'c')] = ['epsilon']
parsing_table[("B'", 'b')] = ['epsilon']
parsing_table[("B'", '$')] = ['epsilon']
parsing_table[('C', 'c')] = ['c']
# Print the parsing table
for key, value in parsing_table.items():
    print(key, '->', value)
```

```
('S', 'a') -> ['A', 'B']
('A', 'a') -> ['a', "A'"]
('A', 'b') -> ['epsilon']
("A'", 'epsilon') -> ['epsilon']
("A'", 'a') -> ['a', "A'"]
('B', 'b') -> ['b', "B'"]
("B'", 'c') -> ['epsilon']
("B'", 'b') -> ['epsilon']
("B'", '$') -> ['epsilon']
('C', 'c') -> ['c']
```

0s completed at 12:26 PM