Code review

https://github.com/Gabarsolon/FLCD-Parser/tree/main

- Code clarity: break down complex expressions to enhance readability.

example:

```
result =
[self.closure([AnalysisElement(self.productions_for_a_given_non_terminal(self.start_symbol)[0], 0)])]
```

changed to

```
production_for_non_terminal = self.productions_for_a_given_non_terminal(self.start_symbol)
first_element_of_production = production_for_non_terminal[0]
analysis_element = AnalysisElement(first_element_of_production, 0)
result = [self.closure([analysis_element])]
```

- Functionality:

The augmentation of the grammar(addition of a new starting symbol S') is done directly in the grammar input file when it should instead be done in code.

- Separation of concerns, modularity, and readability

A separate Grammar class and a separate LR0 class would be more modular and maintainable.

The Grammar class can focus on representing the grammar, reading it from a file, and providing information about the grammar, such as productions and non-terminals.

The LR0 class can handle the LR(0) parsing-related functionalities, such as computing closure, constructing the canonical collection of LR(0) items, and building the parsing table.

- Conciseness and readability

Consider converting the AnalysisElement class into a data class using the @dataclass decorator. This would eliminate the need to write the __init__, __eq__ methods manually, resulting in more concise and readable code. Additionally, it takes advantage of Python's built-in data class functionality.

- Consistent naming

Stick to one convention for naming functions and variables. For example, some functions are named using snake case (read_sequence_from_file), while others use camel case (generateOutputTree).

- Minimize object instantiation

In the ‘main’ function, a new Grammar object is created inside the loop. It might be more efficient to reuse the existing grammar object initialized outside the loop, especially if the grammar doesn't change.

- Comments

In the case of some more complex functions, there was a lack of comments, necessary to better explain the purpose of specific blocks of code, so the person reading could figure its meaning faster.