https://github.com/roxanazachman01/FLCD

Documentation:

Statement:

Implement a parser algorithm (cont.) - as assigned by the coordinating teacher, at the previous lab (PARSER LL(1))

Remark: Lab work evaluation is not done per project/team, but per team member (reflecting the individual contribution to what has been delivered). Please make sure that you split the tasks in a balanced way among team members! In case you decide to do pair programming, each team member is supposed to know all the details of what has been implemented!

Deliverables:

- Functions corresponding to the assigned parsing strategy + appropriate tests, as detailed below:
- LL(1) - functions FIRST, FOLLOW

Implementation:

FIRST:

The first table is implemented as a list of maps, where the key is a TermOrNonTerm and the value is a set of terminals. Each element of the list corresponds to an iteration in the algorithm for constructing the first table. The result, the computed first, is the last element of the list, a map of the same form.

The first table is initialized by adding the initial map to the list. In this map, it adds for each terminal its value and for each nonterminal it adds the first element in the right hand-side of its productions, if that element is a terminal.

The algorithm consists of a do while which stops when the last 2 elements of the lists are equal. Each step of the do-while represents a new iteration for the first table. We start by initialising the new iteration with the values from the previous iteration. Then, for each nonterminal, we get all its productions. For each of its productions, we go further only if all the elements in their right hand-side have the first value computed for the previous iteration. If that’s the case, we get the first element of the right hand-side. If it is a terminal, we simply add it to the set of terminals of that nonterminal. If not, then we need to check the non-terminals in the right hand-side until we find a nonterminal that contains epsilon in the previous iteration. If all the non-terminals from the right hand-side contain epsilon in the previous iteration, then we need to also add epsilon. If not, then we should just add the values from the previous iteration for each of them, until we find one that does not contain epsilon, and make sure we don't add the epsilon. (example: S->AB; if previous iteration of A contains epsilon, then we need to also check previous iteration of B. If that doesn't contain epsilon, then we should make sure first of S does not contain it. Else, if all the non-terminals contain epsilon, then we should add it to S).

The algorithm ends when 2 consecutive iterations have computed the same values. The result, first, is the last iteration.

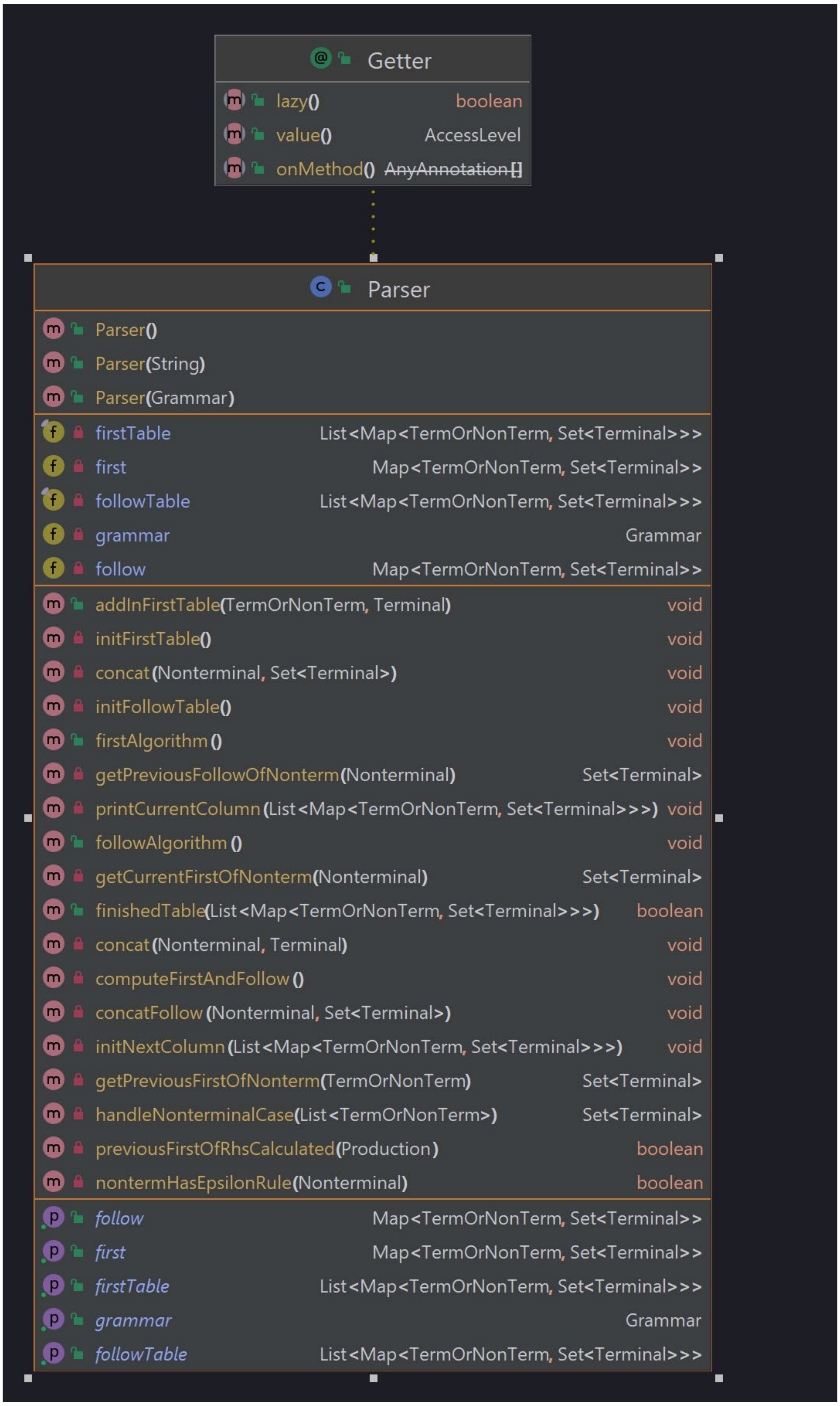
FOLLOW:

The follow table is implemented as a list of maps, where the key is a TermOrNonTerm and the value is a set of terminals. Each element of the list corresponds to an iteration in the algorithm for constructing the first table. The result, the computed follow, is the last element of the list, a map of the same form.

The follow table is initialized by adding the initial map to the list. In this map, it adds for each terminal its value and for each nonterminal it adds the first element in the right hand-side of its productions, if that element is a terminal.

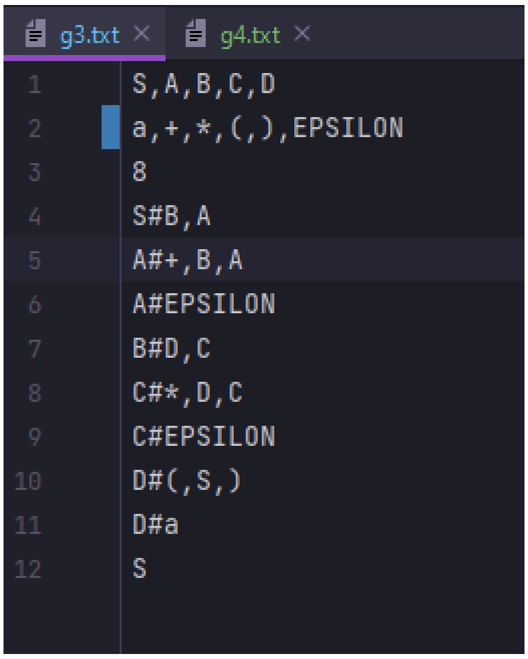
The algorithm consists of a do while which stops when the last 2 elements of the lists are equal. Each step of the do-while represents a new iteration for the follow table. We start by initialising the new iteration with the values from the previous iteration. Then, for each nonterminal B, we get all the productions that contain B in the right hand-side. Denote the left hand-side of those production with A. Then, we need to get the element that comes after B in that right hand-side. If B was the last element, then we add to the follow of B the follow of A in the previous iteration. If not, get that element. If the first of that element does not contain epsilon, then we add to the follow of B the first of that element. If not, we add to the follow of B the first of that element and also the follow of A, but without epsilon.

The algorithm ends when 2 consecutive iterations have computed the same values. The result, follow, is the last iteration.

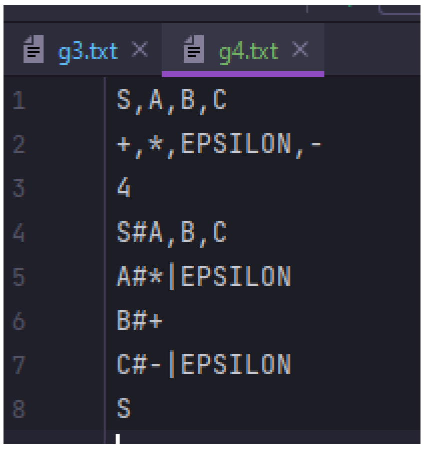

Getter
lazy() boolean
value() AccessLevel
onMethod() ~~AnyAnnotation[]~~
Parser
Parser()
Parser(String)
Parser(Grammar)
firstTable List<Map<TermOrNonTerm, Set<Terminal>>>
first Map<TermOrNonTerm, Set<Terminal>>
followTable List<Map<TermOrNonTerm, Set<Terminal>>>
grammar Grammar
follow Map<TermOrNonTerm, Set<Terminal>>
addInFirstTable(TermOrNonTerm, Terminal) void
initFirstTable() void
concat(Nonterminal, Set<Terminal>) void
initFollowTable() void
firstAlgorithm() void
getPreviousFollowOfNonterm(Nonterminal) Set<Terminal>
printCurrentColumn(List<Map<TermOrNonTerm, Set<Terminal>>>) void
followAlgorithm() void
getCurrentFirstOfNonterm(Nonterminal) Set<Terminal>
finishedTable(List<Map<TermOrNonTerm, Set<Terminal>>>) boolean
concat(Nonterminal, Terminal) void
computeFirstAndFollow() void
concatFollow(Nonterminal, Set<Terminal>) void
initNextColumn(List<Map<TermOrNonTerm, Set<Terminal>>>) void
getPreviousFirstOfNonterm(TermOrNonTerm) Set<Terminal>
handleNonterminalCase(List<TermOrNonTerm>) Set<Terminal>
previousFirstOfRhsCalculated(Production) boolean
nontermHasEpsilonRule(Nonterminal) boolean
follow Map<TermOrNonTerm, Set<Terminal>>
first Map<TermOrNonTerm, Set<Terminal>>
firstTable List<Map<TermOrNonTerm, Set<Terminal>>>
grammar Grammar
followTable List<Map<TermOrNonTerm, Set<Terminal>>>

Testing: tests the parser computes first and follow correctly for 2 different input files: one with a grammar from the seminar and one with a custom simple grammar to check the the epsilon case

Input file:

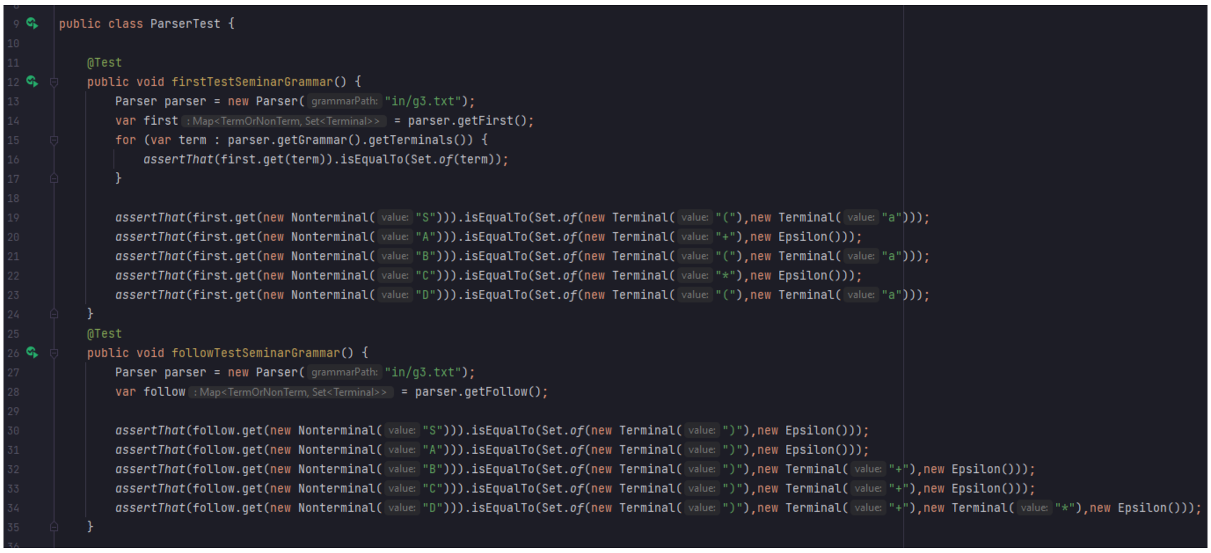


```
S,A,B,C,D
a,+,*,(,),EPSILON
8
S#B,A
A#+,B,A
A#EPSILON
B#D,C
C#*,D,C
C#EPSILON
D#(,S,)
D#a
S
```



```
S,A,B,C
+,*,EPSILON,-
4
S#A,B,C
A#*|EPSILON
B#+
C#-|EPSILON
S
```

Tests:

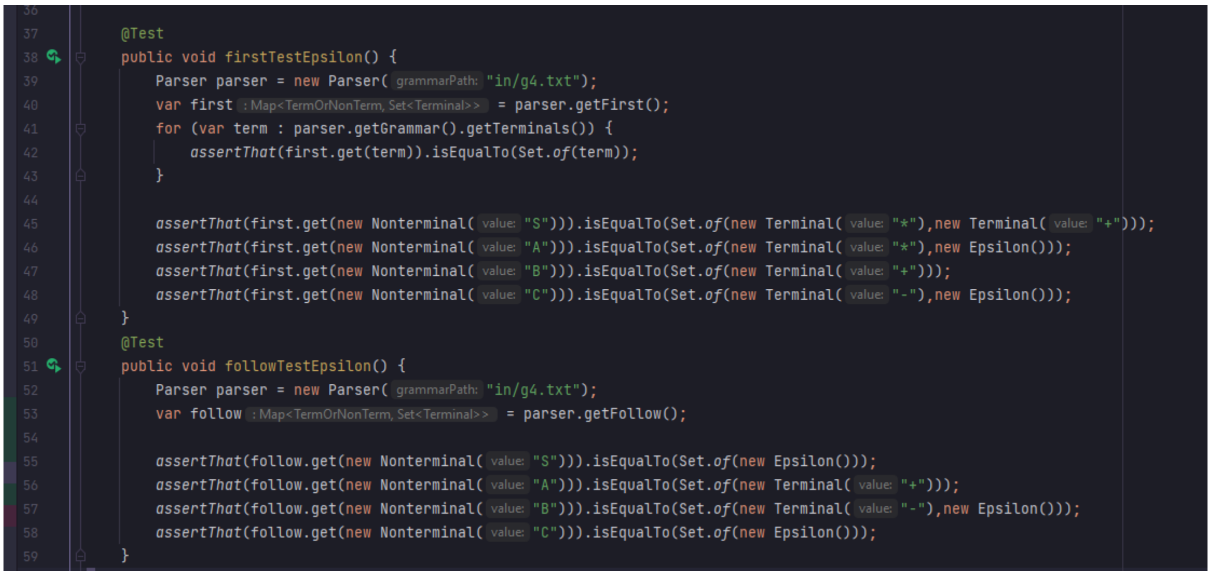


```java
public class ParserTest {

    @Test
    public void firstTestSeminarGrammar() {
        Parser parser = new Parser( grammarPath: "in/g3.txt");
        var first : Map<TermOrNonTerm, Set<Terminal>> = parser.getFirst();
        for (var term : parser.getGrammar().getTerminals()) {
            assertThat(first.get(term)).isEqualTo(Set.of(term));
        }

        assertThat(first.get(new Nonterminal( value: "S"))).isEqualTo(Set.of(new Terminal( value: "("),new Terminal( value: "a")));
        assertThat(first.get(new Nonterminal( value: "A"))).isEqualTo(Set.of(new Terminal( value: "+"),new Epsilon()));
        assertThat(first.get(new Nonterminal( value: "B"))).isEqualTo(Set.of(new Terminal( value: "("),new Terminal( value: "a")));
        assertThat(first.get(new Nonterminal( value: "C"))).isEqualTo(Set.of(new Terminal( value: "*"),new Epsilon()));
        assertThat(first.get(new Nonterminal( value: "D"))).isEqualTo(Set.of(new Terminal( value: "("),new Terminal( value: "a")));
    }
    @Test
    public void followTestSeminarGrammar() {
        Parser parser = new Parser( grammarPath: "in/g3.txt");
        var follow : Map<TermOrNonTerm, Set<Terminal>> = parser.getFollow();

        assertThat(follow.get(new Nonterminal( value: "S"))).isEqualTo(Set.of(new Terminal( value: ")"),new Epsilon()));
        assertThat(follow.get(new Nonterminal( value: "A"))).isEqualTo(Set.of(new Terminal( value: ")"),new Epsilon()));
        assertThat(follow.get(new Nonterminal( value: "B"))).isEqualTo(Set.of(new Terminal( value: ")"),new Terminal( value: "+"),new Epsilon()));
        assertThat(follow.get(new Nonterminal( value: "C"))).isEqualTo(Set.of(new Terminal( value: ")"),new Terminal( value: "+"),new Epsilon()));
        assertThat(follow.get(new Nonterminal( value: "D"))).isEqualTo(Set.of(new Terminal( value: ")"),new Terminal( value: "+"),new Terminal( value: "*"),new Epsilon()));
    }
```



```java
    @Test
    public void firstTestEpsilon() {
        Parser parser = new Parser( grammarPath: "in/g4.txt");
        var first : Map<TermOrNonTerm, Set<Terminal>> = parser.getFirst();
        for (var term : parser.getGrammar().getTerminals()) {
            assertThat(first.get(term)).isEqualTo(Set.of(term));
        }

        assertThat(first.get(new Nonterminal( value: "S"))).isEqualTo(Set.of(new Terminal( value: "*"),new Terminal( value: "+")));
        assertThat(first.get(new Nonterminal( value: "A"))).isEqualTo(Set.of(new Terminal( value: "*"),new Epsilon()));
        assertThat(first.get(new Nonterminal( value: "B"))).isEqualTo(Set.of(new Terminal( value: "+")));
        assertThat(first.get(new Nonterminal( value: "C"))).isEqualTo(Set.of(new Terminal( value: "-"),new Epsilon()));
    }
    @Test
    public void followTestEpsilon() {
        Parser parser = new Parser( grammarPath: "in/g4.txt");
        var follow : Map<TermOrNonTerm, Set<Terminal>> = parser.getFollow();

        assertThat(follow.get(new Nonterminal( value: "S"))).isEqualTo(Set.of(new Epsilon()));
        assertThat(follow.get(new Nonterminal( value: "A"))).isEqualTo(Set.of(new Terminal( value: "+")));
        assertThat(follow.get(new Nonterminal( value: "B"))).isEqualTo(Set.of(new Terminal( value: "-"),new Epsilon()));
        assertThat(follow.get(new Nonterminal( value: "C"))).isEqualTo(Set.of(new Epsilon()));
    }
```