

Lab 5 FLCD - Parser

Pop Daniel Avram + Popa Alex Ovidiu

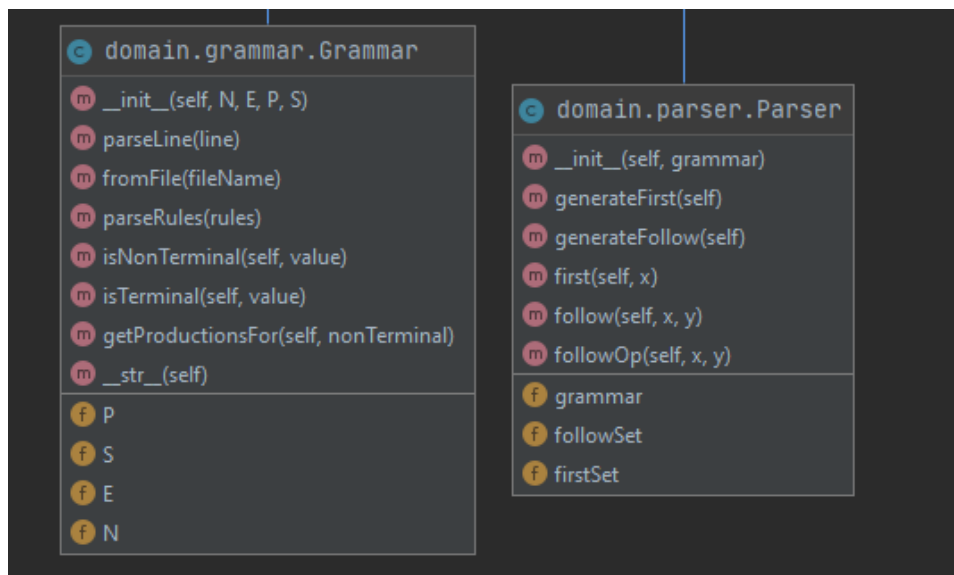
936/1

<https://github.com/alexovidiupopa/flcd/tree/main/parser>

The Grammar class has a field for each (N, E, P, S) set of the grammar, namely terminals, non terminals, productions and starting symbols.

The set of productions P is kept as a list of tuples, of the type (startingSymbol, dest), both strings.

In the Grammar class, most of the methods are for file parsing, however getProductionsFor returns a list for all productions for the specific nonTerminal, for example (S, aA), (S, Epsilon).



Since we are implementing the LL(1) algorithm, we also implemented the first and follow algorithms.

The first algorithm builds a set for each non-terminal that contains all terminals from which we can start a sequence, starting from that given non-terminal.

The follow builds a set for each non-terminal basically returns the “first of what’s after”, namely all the non-terminals into which we can proceed from the given nonterminal.

Having these 2 sets built for each non-terminal (and for terminals also, but those are trivial), we proceed to build the LL(1) parse table. We follow the rules given in the lecture: we build a table that has as rows all non-terminals + terminals, and as rows, all terminals, plus the “\$” sign in both rows and columns. We then follow the rules given in Ms. Motogna’s seventh lecture, slide 9.

Having the parse table built, the next step is parsing a given input sequence with the LL(1) parsing algorithm, following the push/pop rules from slide 13 in same lecture 7. This will build an output which we subsequently recursively build a tree in a depth first search manner – we start from the root, and then we take care of its first child and then right sibling.

The last step is iterating through the tree in a breadth first manner and printing the obtained data.