G == (Vt, Vn, <prog>, R)

Vt = {PROGRAMM, END, ;, ⊥, ℰ, ", ', true, false, integer, bool, string, ,, || , &&,==,!=,=, <, >, ++, +, (, ),/,\*, -, \_, {, }, for, while, if, else, do, endwhile, endif, endfor, пробел, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, }

Vn = {<prog>, <block>, <decls>, <decl>, <literal>, <strLiteral>, <numLiteral>, <boolLiteral>, <type>, <id>, <symbol>, <seqSymbol>, <specSumbol>, <letter>, <seq>, <digit>, <dSeq>, <nDigit>, <stmts>, <stmt>, <assEx>, <orEx>, <addEx>, <equalEx>, <relEx>, <expr>, <addEx> <multiEx>, <unaryEx>, <postfixEx>, <primaryEx>, <matched\_stmt>, <open\_stmt>, <orExAD>, <andExAD>, <equalExAD>, <addExAD>, <multiExAD>, <open\_stmtAD>}

R = {

<prog> ::= PROGRAMM;<decls><stmts>END;⊥,

<block> ::= {<decls><stmts>},

<decls> ::= <decl><decls> | ℰ,

<decl> ::= <type><id>;,

<literal> ::= <numLiteral> | <strLiteral> | <boolLiteral>,

<strLiteral> ::= "<seqSymbol>"|'<Symbol>',

<numLiteral> ::= <digit> | <nDigit><dSeq>,

<boolLiteral> ::= true | false,

<type> ::= integer | bool | string,

<id> ::= <letter> | <letter><seq>,

<symbol> ::= <letter> | <digit> | <specSymbol>,

<seqSymbol> ::= <symbol> <seqSymbol> | <symbol>,

<specSymbol> ::= , | " | ' | ; | / | \* | = | < | > | ++ | { | } | + | ( | ) | - | ℰ | || | && | == | !=,

<letter> ::= \_ | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z,

<seq> ::= <letter> | <letter><seq> | <digit> | <digit><seq>,

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9,

<dSeq>::= <digit><dSeq> | <digit>,

<nDigit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9,

<stmts> ::= <stmt><stmts> | ℰ,

<stmt> ::= <stmtAD> | <decls><stmtAD>,

<stmtAD> ::= <assEx>; | <block> | for(<assEx>; <orEx>; <expr>) <stmt> endfor; | <expr>; | while(<orEx>) <stmt> endwhile; | if (<orEx>) <stmt> else <stmt> endif; | if (<orEx>) <stmt> endif; | ℰ,

<assEx> ::= <id> = <orEx> | ℰ,

<orEx> ::= <andEx><orExAD>,

<orExAD> ::= ||<andEx><orExAD> | ℰ,

<andEx> ::= <equalEx><andExAD>,

<andExAD> ::= && <equalEx><andExAD> | ℰ,

<equalEx> ::= <relEx><equalExAD>,

<equalExAD> ::= == <relEx><equalExAD> | != <relEx><equalExAD> | ℰ,

<relEx>::= <addEx> < <addEx> | <addEx> > <addEx> | <addEx>,

<expr> ::= <assEx>,<expr> | <assEx> ,

<addEx>::= <multiEx><addExAD>,

<addExAD> ::= + <multiEx><addExAD> | - <multiEx><addExAD> | ℰ,

<multiEx> ::= <unaryEx><multiExAD>,

<multiExAD> ::= \* <unaryEx><multiExAD> | / <unaryEx><multiExAD> | ℰ,

<unaryEx> ::= ++<postfixEx> | -<postfixEx> | <postfixEx>,

<postfixEx> ::= <primaryEx>++ | <primaryEx>,

<primaryEx> ::= (<orEx>) | <id> | <literal>.

}