Grammar Rules for TerminaScript Programming lanugage

prog : [expr]

expr : LET ID EQ arith | comp : IF (comp {AND/OR comp}) { prog } : FOR (ID ASSIGN INT ARROW INT) { prog } : FUNC ID({arith}) { prog } : comparison

comparison : arith {==,!=,>,>=,<,<=} comparison

arith : term {(+/-) term}

term : factor {(* / /) factor}

factor : INT : STRING : ID : (expr) : - factor : ID({arith})

comp : (NOT) comp : arith (EE|GE|LE|GTE|LTE) arith

prog : expr*

expr : {KEYWORD:VAR} IDENTIFIER EQ expr : comp (AND | OR) comp : IDENTIFIER(expr,expr)

comp : (NOT) comp : arith ((EE|LT|GT|LTE|GTE) arith)

arith : term ((PLUS | MINUS)) term)*/def

term : factor ((MUL | DIVICE) factor)*

factor : PLUS | MINUS factor

power : call (POW factor)

call : atom (expr (COMMA expr)*)

atom : INT|FLOAT|INDENTIFIER : LPAREN expr RPAREN : if-expr : func-def : for-expr

if-expr : if expr then expr : elif expr then expr : else expr

for-expr : for IDENTIFIER EQ expr to expr : step ? : prog NEXT

if { consequences [{ "condition":BinaryOpNode "consequence":ProgramNode }, { "condition":BinaryOpNode "consequence":ProgramNode }, { "condition":BinaryOpNode "consequence":ProgramNode },] alternative ProgramNode }