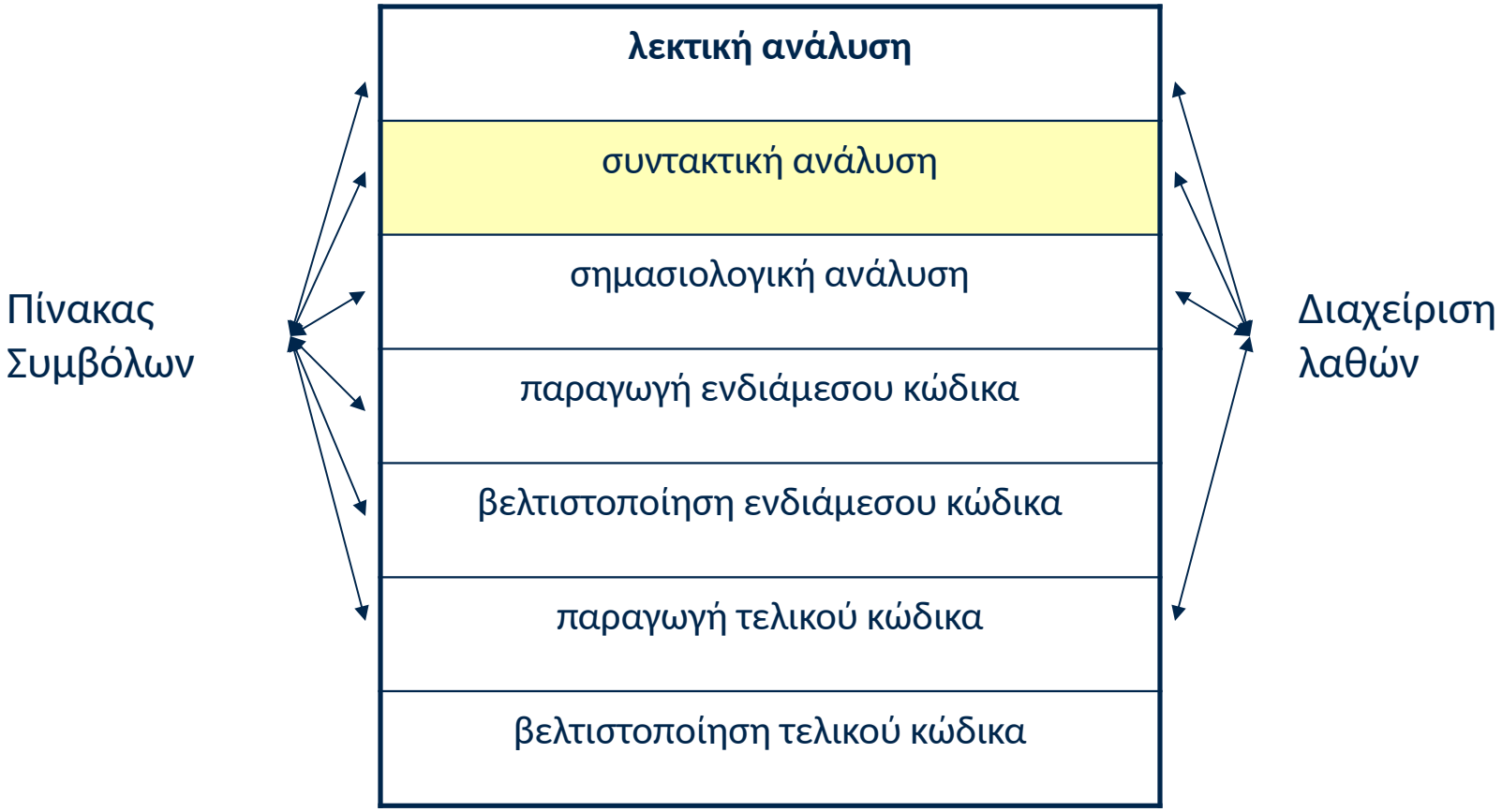
Συντακτικός Αναλυτής

Διαλέξεις στο μάθημα: Μεταφραστές
Γεώργιος Μανής

Οι Φάσεις της Μεταγλώττισης



Συντακτική Ανάλυση

Λειτουργία Συντακτικού Αναλυτή

- Γίνεται έλεγχος για να διαπιστωθεί εάν το πηγαίο πρόγραμμα **ανήκει ή όχι** στη γλώσσα
- δημιουργεί το **κατάλληλο «περιβάλλον»** μέσα από το οποίο αργότερα θα κληθούν οι σημαντικές ρουτίνες.
- Υπάρχουν πολλοί τρόποι για να κατασκευαστεί ένας συντακτικός αναλυτής
- Θα προτιμήσουμε τη συντακτική ανάλυση με **αναδρομική κατάβαση**
- Βασίζεται σε **γραμματική LL(1)**

Γραμματική LL(1)

- L : left to right
- L : leftmost derivation
- (1) : one look-ahead symbol

  - Η γραμματική LL(1) αναγνωρίζει από **αριστερά στα δεξιά**, **την αριστερότερη δυνατή παραγωγή** και όταν βρίσκεται σε δίλλημα ποιον κανόνα να ακολουθήσει της αρκεί να κοιτάξει το **αμέσως επόμενο σύμβολο** στην συμβολοσειρά εισόδου

Γραμματική LL(1)

Παράδειγμα:

```
S ::=   while(condition) S
S ::=   print(expression)
S ::=   input(id)
S ::=   { S }
```

στο παραπάνω τμήμα γραμματικής, όταν πρέπει να αναγνωρίζουμε S, τότε

- ακολουθούμε τον πρώτο κανόνα αν η επόμενη λεκτική μονάδα στη είσοδο είναι το while,
- τον δεύτερο εάν είναι το print,
- τον τρίτο εάν είναι το input
- και τον τέταρτο εάν είναι το άνοιγμα αγκίστρου

Γραμματική LL(1)

Παράδειγμα:

```
# if statement
ifStat    →    if ( condition )
                   statements(1)
               elsepart
# else part is optional
elsepart  →    else
                   statements(2)
          |    ε
```

Το *elsepart* υλοποιεί προαιρετικό τμήμα της εντολής

όταν τελειώσουν τα statements, αν ακολουθεί *else* ενεργοποιείται το πρώτο τμήμα του κανόνα *elsepart* αλλιώς το δεύτερο που δίνει το κενό

Γραμματική LL(1)

Παράδειγμα:

```
# term in arithmetic expression
term          →    factor
                   ( MUL_OP factor )*
```

Το *term* αντικαθίσταται με το *factor* και στη συνέχεια ακολουθούν κανένας η περισσότεροι *factor* χωρισμένοι με πολλαπλασιαστικούς τελεστές

Γραμματική LL(1)

Παράδειγμα:

```
# factor in boolean  expression
boolfactor    →    not [ condition ]
              |    [ condition ]
              |    expression REL_OP expression
```

επιλογή 1: ξεκινάει με *not*

επιλογή 2: ξεκινάει με *[*

επιλογή 3: οτιδήποτε άλλο

Σχέδιο Κώδικα

Το σχέδιο κώδικα ενός συντακτικού αναλυτή μπορεί να είναι κάπως έτσι:

```
def syntax_analyzer():
    global token
    token = self.get_token()
    self.program()
    print('compilation successfully completed')
```

Εσωτερική Λειτουργία

- Για κάθε έναν από τους **κανόνες** της γραμματικής, φτιάχνουμε και ένα αντίστοιχο **υποπρόγραμμα**
- Όταν συναντάμε **μη τερματικό** σύμβολο **καλούμε** το αντίστοιχο υποπρόγραμμα
- Όταν συναντάμε **τερματικό** σύμβολο, τότε
  - εάν και ο λεκτικός αναλυτής **επιστρέφει λεκτική μονάδα που αντιστοιχεί** στο τερματικό αυτό σύμβολο έχουμε αναγνωρίσει **επιτυχώς** τη λεκτική μονάδα
  - αντίθετα εάν ο λεκτικός αναλυτής **δεν επιστρέψει τη λεκτική μονάδα που περιμένει** ο συντακτικός αναλυτής, έχουμε **λάθος** και καλείται ο διαχειριστής σφαλμάτων
- **Όταν αναγνωριστεί και η τελευταία λέξη** του πηγαίου προγράμματος, τότε η συντακτική ανάλυση έχει στεφτεί με **επιτυχία**.

Παράδειγμα – Ένα Τμήμα Γραμματικής

```
<PROGRAM>    ::= program ID
                              <PROGRAMBLOCK>

<PROGRAMBLOCK>        ::= <DECLARATIONS>
                              <SUBPROGRAMS>
                              <BLOCK>

<BLOCK>               ::= begin
                              <SEQUENCE>
                          end

<DECLARATIONS>        ::= <CONSTDECL>
                              <VARDECL>

<CONSTDECL> ::= const <ASSIGNLIST> ; | ε

    ...
```

Από τη Γραμματική στον Κώδικα

<PROGRAM> ::= **program** ID <PROGRAMBLOCK>

```
void program()
{
        if (token==programtk) {
                token=lex();
                if (token==idtk) {
                        token=lex();
                        programBlock();  }
                else error("program name exprected");      }
        else error ("the keyword 'program' was expected");
        }
}
```

Από τη Γραμματική στον Κώδικα

<PROGRAMBLOCK> ::= <DECLARATIONS>
<SUBPROGRAMS>
<BLOCK>

```
void programBlock()
{
   declarations();
   subprograms();
   block();
}
```

Από τη Γραμματική στον Κώδικα

<BLOCK> ::= **begin** <SEQUENCE> **end**

```
void block()
{
        if (token==begintk)
        {
                token=lex();
                sequence();
                if (token==endtk)
                        token=lex();
                else error ("the keyword 'end' was expected");
        }
        else error ("the keyword 'begin' was expected");
}
```

Από τη Γραμματική στον Κώδικα

<DECLARATRIONS> ::= <CONSTDECL><VARDECL>

```
void declarations()
{
        constdecl();
        vardecl();
}
```

<CONSTDECL>::= **const** <ASSIGNLIST> | ε

```
void constdecl()
{
if (token==consttk) {
        token=lex();
        assignlist();
}
 }
```

Παράδειγμα – Ένα Τμήμα Γραμματικής

```
<IF-STAT>    ::= if <CONDITION>
                          then  <BLOCK>
                          <ELSEPART>

<ELSEPART>           ::= ε | else <BLOCK>

    ...

<BOOLFACTOR>         ::= not <CONDITION> |
                          ( <CONDITION> ) |
                          <EXPRESSION>
                          <RELATIONAL-OPER>
                                  <EXPRESSION>

<EXPRESSION>         := <OPTIONAL-SIGN> <TERM>
                          ( <ADD-OPER> <TERM>)*

    ...
```

Από τη Γραμματική στον Κώδικα

<IF-STAT> ::= **if** <CONDITION> **then** <BLOCK> <ELSEPART>

```
void if_stat()
{
        if (token==iftk)
        {
                token=lex();
                condition();
                if (token==thentk)
                {       token=lex();
                        block();
                        elsepart();
                }
                else error ("the keyword 'then' was
expected");
        }
        else error ("the keyword 'if' was expected");
}
```

θα υπάρχει πάντα
if αλλιώς δε θα μπει
μέσα στην IF-STAT

Από τη Γραμματική στον Κώδικα

<ELSEPART> ::= **else** <BLOCK> | ε

```
void elsepart()
{
        if (token==elsetk)
        {
                token=lex();
                block();

        }
}
```

Από τη Γραμματική στον Κώδικα

<BOOLFACTOR> ::= **not** <CONDITION> | **[** <CONDITION> **]** |
<EXPRESSION> <RELATIONAL-OPER> <EXPRESSION>

```
void boolFactor()
{
        if (token==nottk) {
                token=lex();
                condition(); }
        else if (token==leftsquarebrackettk) {
                token=lex();
                condition();
                if (token==rightsquarebrackettk)
                        token=lex()
                else error("right square bracket expected"); }
        else {
                expression();
                relationalOper();
                expression(); }
}
```

Από τη Γραμματική στον Κώδικα

<EXPRESSION>::=<OPTIONAL-SIGN> <TERM>

(<ADD-OPER> <TERM>) *

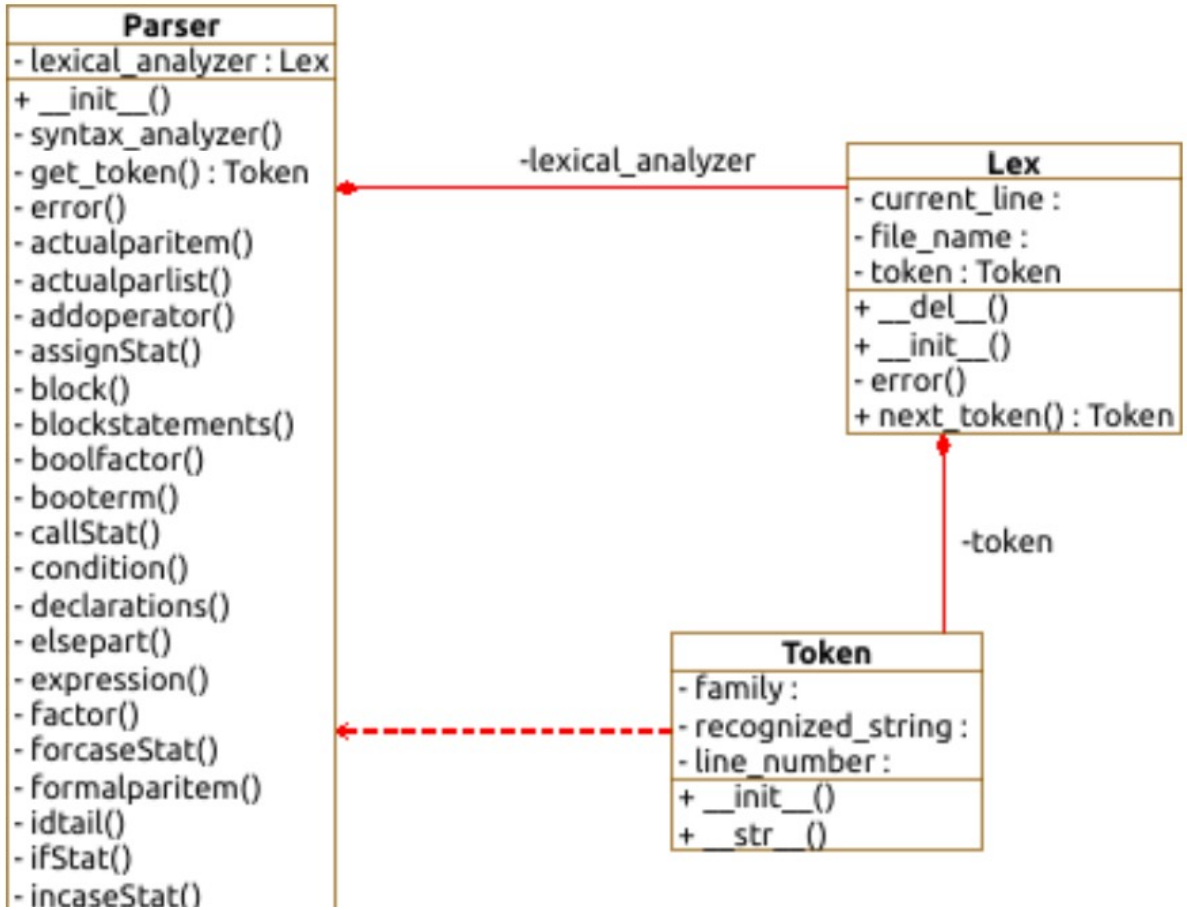
```
void expression()
{
        optionalSign();
        term();
        while (token==plustk || token==minustk)
        {
                addOper();
                term();
        }
}
```

Από τη Γραμματική στον Κώδικα

```
def program():
    global token
    if token.recognized_string == 'program':
        token = self.get_token()
        if token.family == 'id':
            token = self.get_token()
            self.block()
            if token.recognized_string == '.':
                token = self.get_token()
                    if token.recognized_string == 'eof':
                        token = self.get_token()
                    else:
                        self.error(...)
            else:
                self.error(...)
        else:
            self.error(...)
    else:
        self.error(...)
```

```
# "program" is the starting symbol
# followed by its name and a block
# Every program ends with a fullstop
program  →   program ID
             block
             .
```

Ο Συντακτικός Αναλυτής στο Διάγραμμα Κλάσεων



Συντακτική Ανάλυση με το Εργαλείο yacc

Παράδειγμα Γραμματικής

```
E -> E + E
E -> E * E
E -> id
```

```
E -> E * E          (r2)
  -> E * z          (r3)
  -> E + E * z      (r1)
  -> E + y * z      (r3)
  -> x + y * z      (r3)
```

Παράδειγμα calculator - lex

```
%token INTEGER

%{
#include <stdlib.h>
void yyerror(char *);
#include "y.tab.h"
%}

%%

[0-9]+      {
                yylval = atoi(yytext);
                return INTEGER;
            }

[-+\n]      return *yytext;

[ \t]       ; /* skip whitespace */

.           yyerror("invalid character");

%%
```

Παράδειγμα calculator - yacc

```
%{
    int yylex(void);
    void yyerror(char *);
%}

%token INTEGER

%%

program:
        program expr '\n'
        |
        ;

expr:
        INTEGER
        | expr '+' expr
        | expr '-' expr
        ;

%%
```

ευχαριστώ !