

# Αρχές Γλωσσών Προγραμματισμού και Μεταφραστών: Εργαστηριακή Άσκηση 2011-2012

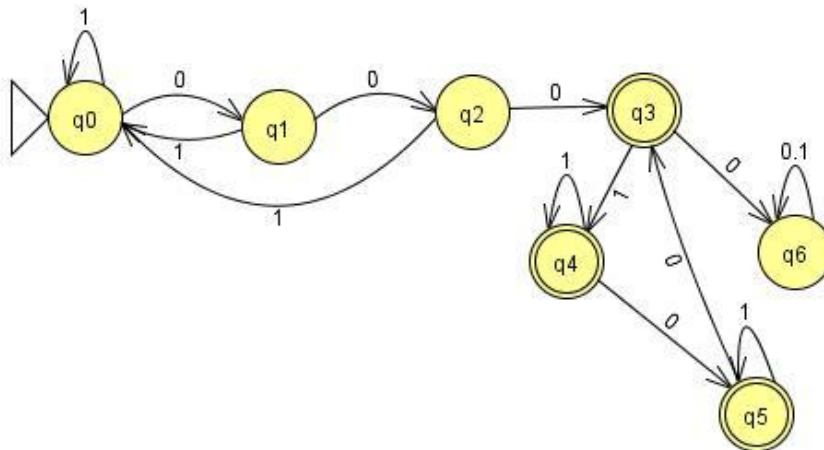
## Συνεργάτες:

1. Μονοπάτης Δημήτριος 4776
2. Σταυρουλάκης Αλέξανδρος 5076
3. Χατζηαντωνίου Παναγιώτης 5102
4. Τσώνη Σοφία 5144

## Θεωρητικό Τμήμα

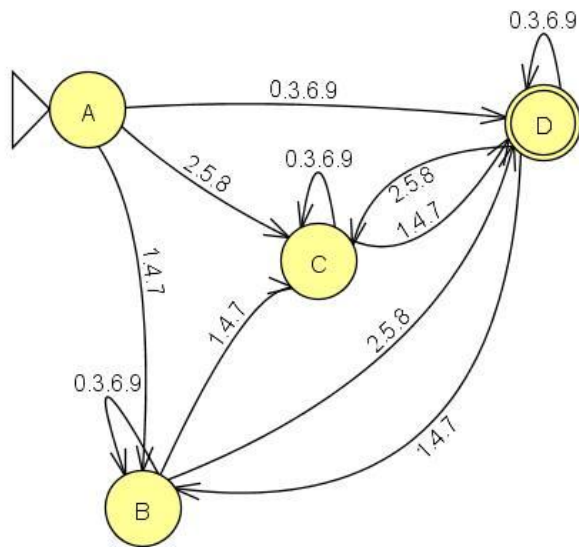
### Ζήτημα 1 Αυτόματα και Κανονικές Γλώσσες

1) Ντετερμινιστικό πεπερασμένο αυτόματο:



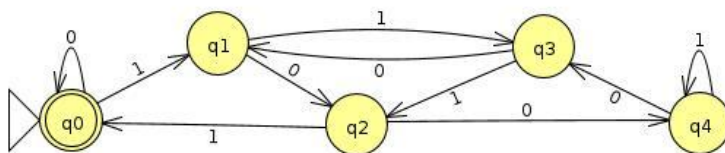
Κανονική έκφραση:  $1^*(000)(1^*0|001^*)^*$

2) Ντετερμινιστικό πεπερασμένο αυτόματο:



3) Κανονική έκφραση:  $b(b^*a?(aa)^+)*b^+$

Bonus



## Ζήτημα 2 Γραμματικές

1) Η δοθείσα γραμματική τροποποιήθηκε όπως φαίνεται παρακάτω:

```
<decl1> ::= <type> <id> (<formals>) {<stmt1>*}
<stmt1> ::= <simple> | return <expr>;
           | while(<expr>){<stmt1>*, break; | continue; | e}
           | switch(<expr>){<stmt1>*, break; | e}
<decl2> ::= void <id> (<formals>){<stmt2>*}
<stmt2> ::= <simple> | return;
           | while(<expr>){<stmt2>*, break; | continue; | e}
           | switch(<expr>){<stmt2>*, break; | e}
```

- 2) α)  $First(S) = \{ [, a \}$   
 $First(X) = \{ \epsilon, +, b, - \}$   
 $First(Y) = \{ \epsilon, - \}$

$Follow(S) = \{ \$, +, b, -, ] \}$   
 $Follow(X) = \{ [, c \}$   
 $Follow(Y) = \{ b, c, ] \}$

1.  $Predict( \langle S \rangle \rightarrow [ \langle S \rangle \langle X \rangle ] ) = \{ \$ \}$
2.  $Predict( \langle S \rangle \rightarrow a ) = \{ a \}$
3.  $Predict( \langle X \rangle \rightarrow \epsilon ) = \{ c, \$ \}$
4.  $Predict( \langle X \rangle \rightarrow + \langle S \rangle \langle Y \rangle ) = \{ + \}$
5.  $Predict( \langle X \rangle \rightarrow \langle Y \rangle b ) = \{ -, b \}$
6.  $Predict( \langle Y \rangle \rightarrow \epsilon ) = \{ \$ \}$
7.  $Predict( \langle Y \rangle \rightarrow - \langle S \rangle \langle X \rangle c ) = \{ - \}$

Με βάση τα σύνολα predict που βρήκαμε παραπάνω σχεδιάζουμε τον πίνακα συντακτικής ανάλυσης της γραμματικής:

Μη- τερματικά σύμβολα	Τρέχουσα λεκτική μονάδα εισόδου							
	[	]	+	-	a	b	c	\$\$
$\langle S \rangle$	-	-	-	-	2	-	-	1
$\langle X \rangle$	-	-	4	5	-	5	3	3
$\langle Y \rangle$	-	-	-	7	-	-	-	6

Για string εισόδου a+a-abc έχουμε την παρακάτω top-down συντακτική ανάλυση:

Στοιβά	Είσοδος	Ενέργεια
$\langle S \rangle$	[a+a-abc]	-
$[ \langle S \rangle \langle X \rangle ]$	[a+a-abc]	Predict $\langle S \rangle ::= [ \langle S \rangle \langle X \rangle ]$
$\langle S \rangle \langle X \rangle ]$	a+a-abc]	Match [
a $\langle X \rangle ]$	a+a-abc]	Predict $\langle S \rangle ::= a$
$\langle X \rangle ]$	+a-abc]	Match a
+ $\langle S \rangle \langle Y \rangle ]$	+a-abc]	Predict $\langle X \rangle ::= + \langle S \rangle \langle Y \rangle$
$\langle S \rangle \langle Y \rangle ]$	a-abc]	Match +
a $\langle Y \rangle ]$	a-abc]	Predict $\langle S \rangle ::= a$
$\langle Y \rangle ]$	-abc]	Match a
- $\langle S \rangle \langle X \rangle c ]$	-abc]	Predict $\langle Y \rangle ::= - \langle S \rangle \langle X \rangle c$
$\langle S \rangle \langle X \rangle c ]$	abc]	Match -
a $\langle X \rangle c$	abc]	Predict $\langle Y \rangle ::= a$
$\langle X \rangle c ]$	bc]	Match a
$\langle Y \rangle bc ]$	bc]	Predict $\langle X \rangle ::= \langle Y \rangle b$
εbc]	bc]	Predict $\langle Y \rangle ::= \epsilon$
c]	c]	Match b
]	]	Match c
EOF	ε	Match ]
EOF	EOF	Identification (Αναγνώριση)

Στη top-down ανάλυση υπάρχουν δυο ενέργειες: η πρόβλεψη(predict), όπου προβλέπεται ποιος κανόνας μπορεί να ακολουθήσει κάθε φορά στην ανάλυση και το ταίριασμα(match) κατά το οποίο αναγνωρίζονται τα τερματικά σύμβολα της γραμματικής και αφαιρούνται από τη στοιβά. Όταν το string εισόδου σαρωθεί όλο και όλα τα τερματικά σύμβολα έχουν αναγνωριστεί τότε το string θεωρείται αναγνωρισμένο.

- 3) Στις bottom-up συντακτικές αναλύσεις μπορούν να γίνουν δυο ενεργειες: η ολίσθηση(shift) και η ελάττωση(reduce). Όταν προκύπτει επιλογή ελάττωσης σε κάποιον αρχικό κανόνα της γραμματικής, ενώ ταυτόχρονα δεν έχει σαρωθεί ολόκληρο το string εισόδου (δηλαδή θα έπρεπε να γίνει ολίσθηση) τότε προκύπτει η σύγκρουση ολίσθησης-ελάττωσης.

Για να αντιμετωπιστεί αυτό το πρόβλημα στις LR(k) γραμματικές θα πρέπει το k να είναι μεγαλύτερο του 1, έτσι ώστε να σαρώνονται περισσότεροι χαρακτήρες της εισόδου. Με αυτό τον τρόπο μπορούμε να προβλέψουμε ευκολότερα ποια ενεργεια πρέπει να ακολουθήσει (shift ή reduce), γιατί με περισσότερα σύμβολα η επιλογή ενεργειας είναι προφανέστερη σε σχέση με τη σάρωση ενός χαρακτήρα τη φορά.

Για string εισόδου «db» για τη δοθείσα γραμματική έχουμε την παρακάτω ανάλυση

Στοιβά	είσοδος	ενέργεια
-	db	Shift
d	bEOF	Shift/reduce σύγκρουση (<A>::=d)
db	EOF	Reduce <S>::=db
<S>	EOF	-

Παρατηρούμε ότι όταν σαρωθεί το d μπορεί να γίνει reduce στον κανόνα <A>, αλλά ο κανόνας <A> δεν είναι “αρχικός”\* και επίσης δεν έχει σαρωθεί όλο το string εισόδου, επομένως έχουμε σύγκρουση ολίσθησης/ελάττωσης.

Στις γραμματικές SLR(1) δεν υπάρχουν προβλήματα συγκρούσεων έτσι αφού εμείς εντοπίσαμε σύγκρουση shift/reduce η γραμματική μας είναι LR(1), αλλά όχι SLR(1)

---

\*” Αρχικός” κανόνας είναι ο <S> καθώς σε αυτόν καταλήγει το δέντρο της bottom-up συντακτικής ανάλυσης

# Πρακτικό Τμήμα

## Υλοποίηση Parser της γλώσσας Buzen

Αρχείο flex

```
%{  
  
#include <stdio.h>  
  
#include <stdlib.h>  
  
#include "2012.tab.h"  
  
  
#define YYSTYPE char const *  
  
  
extern int line_no;  
  
extern char txt;  
  
extern char *line;  
  
void ERROR(const char msg[]);  
  
  
%}  
  
  
%%  
  
\n.* { /* Prin kanoyme otidhpote swzoyme thn trexoysa grammh keimenoy, gia ta mhnymata lathoys */  
    if (line != NULL)  
        free(line);  
  
    line = strdup(yytext);  
  
    ++line_no; /* Gia na kseroyme se poia grammh eimaste. */  
  
    yyless(1); /* Stelnoyme pisw sto lex ola ektos toy /n */  
}  
  
"program" {return PROGRAM;}  
  
";" {return SEM;}  
  
"var" {return VAR;}  
  
"integer" {return INTT;}  
  
"boolean" {return BOO;}  
  
"procedure" {return PROCEDURE;}  
  
"function" {return FUN;}  
  
"if" {return IFF;}  
  
"then" {return THENN;}  
  
"else" {return ELSEE;}  
  
"while" {return WHILEE;}
```

```
"do" {return DOO;}

"begin" {return BEGINN;}

"end" {return ENDD;}

"::=" {return ASS;}

"." {return PANKAT;}

"." {return FSTOP;}

"," {return COMMA;}

"(" {return LPAR;}

")" {return RPAR;}

"and" {return ANDD;}

"or" {return ORR;}

"not" {return NOTT;}

"+" {return PLUS;}

"-" {return MINUS;}

"*" {return MULT;}

"div" {return DIVV;}

"mod" {return MODD;}

"=" {return EQUAL;}

"<>" {return DIFF;}

"<" {return SM;}

">" {return BIG;}

"<=" {return SMEQ;}

">=" {return BEQ;}

[0-9]+ {return NUMBER;}

[a-zA-Z][a-zA-Z0-9]* {return ID;}

%%

int yywrap(void) {return 1;}
```

Αρχείο bison

```
%{
/*kathe stoixeio 8a einai sindedemenos ena char*/
#define YYSTYPE char const*

#include <stdio.h>

#include <ctype.h>

    int j=0;

    int ag=0;

    int i;

    char * line = NULL;

    int line_no = 1;

    char txt;

    void yyerror(char const*);

    extern char *yytext;

    extern int yylex();

    FILE *yyin, *yyout;

/* Posa sfalmata eginan */

int errors = 0;

%}

%token PROGRAM

%token SEM

%token VAR

%token INTT

%token BOO

%token PROCEDURE

%token FUN

%token IFF

%token THENN

%token ELSEE

%token WHILEE

%token DOO

%token BEGINN

%token ENDD

%token ASS

%token PANKAT
```

%token FSTOP  
%token COMMA  
%token LPAR  
%token RPAR  
%token ANDD  
%token ORR  
%token NOTT  
%token PLUS  
%token MINUS  
%token MULT  
%token DIVV  
%token MODD  
%token EQUAL  
%token DIFF  
%token SM  
%token BIG  
%token SMEQ  
%token BEQ  
%token ID  
%token NUMBER

%%

/\*grammar rules \*/

prog: PROGRAM id sem block FSTOP;

block: K compoundstatement;

K: localdefinition K | ;

localdefinition: variabledefinition | proceduredefinition |  
functiondefinition ;

variabledefinition: VAR L ;

L: defsomevariables sem L |;

defsomevariables: id M PANKAT datatype;

M: COMMA id M | ;

proceduredefinition: procedureheader block sem;

procedureheader: PROCEDURE id formalparameters sem;

functiondefinition: functionheader block sem;

functionheader: FUN ID formalparameters PANKAT datatype sem;

formalparameters: LPAR formalparameter N RPAR | ;



N: sem formalparameter N | ;  
 formalparameter: id M PANKAT datatype;  
 datatype: INTT | BOO;  
 statement: assignment | ifstatement | whilestatement | procfunccall | compoundstatement;  
 assignment: id ASS expression ;  
 ifstatement: IFF expression THENN statement P;  
 P: ELSEE statement | ;  
 whilestatement: WHILEE expression DOO statement;  
 procfunccall: id LPAR actualparameters RPAR ;  
 actualparameters: expression R |;  
 R: COMMA expression R | ;  
 compoundstatement: BEGINN statement S ENDD ;  
 S: sem statement S | ;  
 expression: unaryoperation expression A | procfunccall A | LPAR expression RPAR A | NUMBER A | id A ;  
 A: binaryoperation A expression | ;  
 binaryoperation: EQUAL | DIFF | ORR | ANDD | SM | BIG | BEQ | SMEQ | PLUS | MINUS | MULT | DIVV | MODD ;  
 unaryoperation: PLUS | MINUS | NOTT ;  
 sem: SEM ;  
 id: ID ;

%%

```

int main(int argc, char **argv){
    //elegxoume an exoume 2o orisma, dld an dothike onoma arxeiou buz
    if(argc==2)
        yyin = fopen( argv[1], "r" );
    else
        yyin = stdin;
    yyout = fopen ( "output", "w" );
    yyparse ();
    if(errors==0)
        printf("Den yphrxe lathos sto arxeio: %s.\n", argv[1]);
    return 0;
}

```

```

void yyerror(char const *s)
{
    printf("line: %d, %s\n", line_no, s);
}

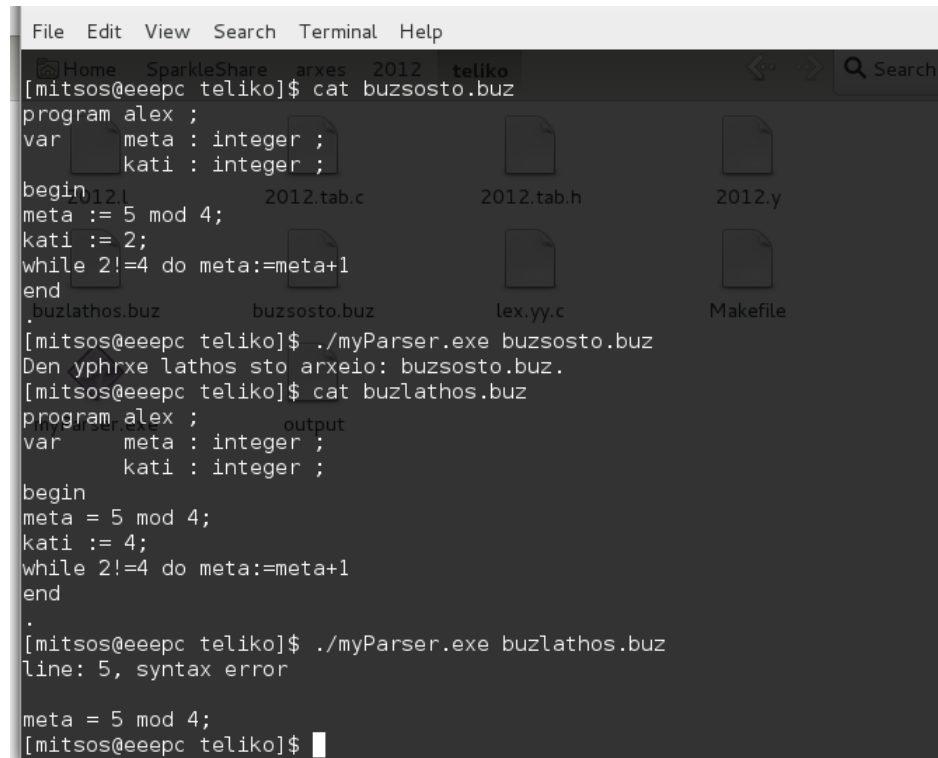
```

```
printf("%s\n", line);

errors++;

}
```

Screenshot από εκτέλεση των αρχείων buzso.to.buz και buzla.thos.buz στον parser



```
File Edit View Search Terminal Help
[mitsos@eeepc teliko]$ cat buzso.to.buz
program alex ;
var      meta : integer ;
        kati : integer ;
begin
meta := 5 mod 4;
kati := 2;
while 2!=4 do meta:=meta+1
end
.
[mitsos@eeepc teliko]$ ./myParser.exe buzso.to.buz
Den yphrxh lathos sto arxeio: buzso.to.buz.
[mitsos@eeepc teliko]$ cat buzla.thos.buz
program alex ;
var      meta : integer ;
        kati : integer ;
begin
meta = 5 mod 4;
kati := 4;
while 2!=4 do meta:=meta+1
end
.
[mitsos@eeepc teliko]$ ./myParser.exe buzla.thos.buz
line: 5, syntax error
meta = 5 mod 4;
[mitsos@eeepc teliko]$
```