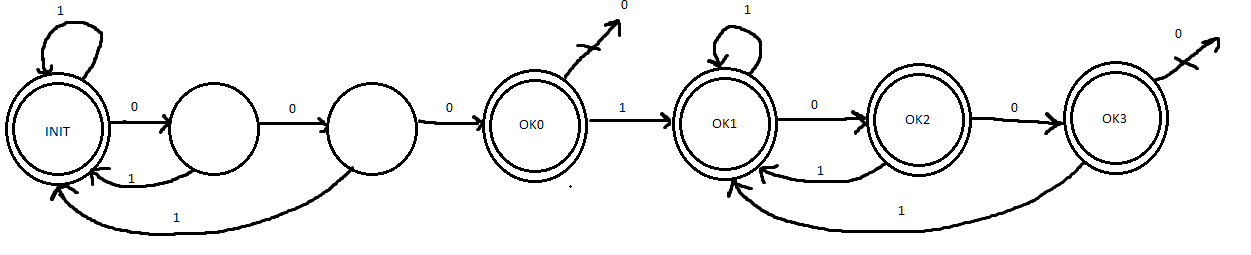
Project μεταφραστές

# Σειμένης Σπύρος 5070 Καλλιβωκάς Δημήτριος 4993

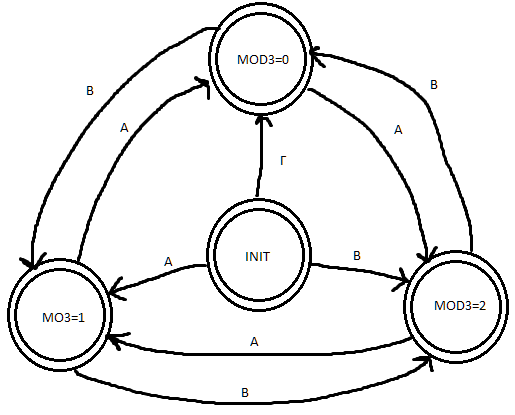
# Ζήτημα 1

**1.**

Ντετερμινιστικό πεπερασμένο αυτόματο 

Κανονική έκφραση:

((0|1)?(0|1)?1)\*000(1(0|1)?(0|1)?)\*

**2**.

A -> 1,4,7

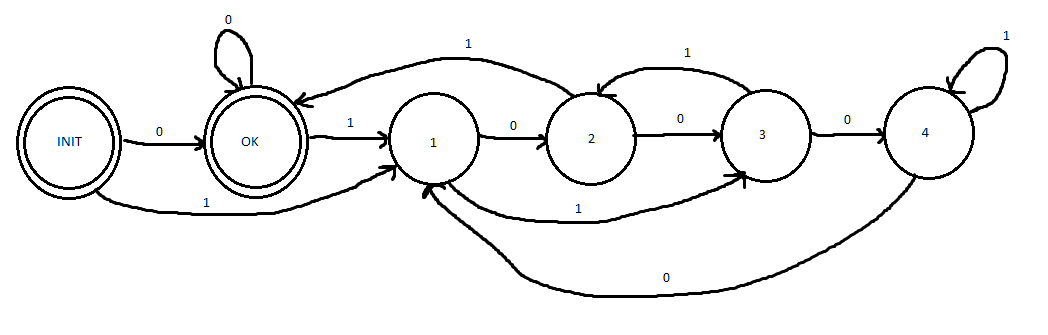
B -> 2,5,8

Γ -> 3,6,9,0

**3**.

b(b\*(aa+)\*)\*b

**4. Παρέχεται και κώδικας σε python που αποδεικνύει ότι λειτουργεί (zitima1-4.py)**

****

# Ζήτημα 2

**1**.

<decl> ::== <type> <id> (<formals>) {<stmt>\*} | void <id>(formals){<stmt\_void\*>}

<stmt> ::== <simple> | return <expr>;   
 |while(<expr>) { (break;|continue;|<stmt>)\* }  
 | switch(<expr>) { ( break; |<stmt>)\* }

<stmt\_void> ::== <simple> | return;   
 |while(<expr>) { (break;|continue;|<stmt\_void>)\* }  
 | switch(<expr>) { ( break; |<stmt\_void>)\* }

**2**.(α)

FIRST(S) = { [,a }  
FIRST(X) = { є,+,-,b }  
FIRST(Y) = { є,- }

FOLLOW(S) = { +,-,b,c,] ,$}  
FOLLOW(X) = { ],c }  
FOLLOW(Y) = { b, c,] }

PREDICT(S = [SX]) = { [ }  
PREDICT(S = a) = { a }  
PREDICT(X= є) = { ],c }  
PREDICT(X = + S Y) = { + }  
PREDICT(X= Y b) = { b, - }  
PREDICT(Y= є) = { b,c,] }  
PREDICT(Y=- S Xc) = { - }

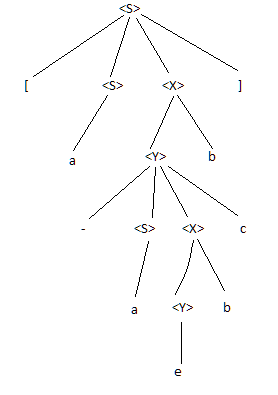
Αυτή η γλώσσα αποτελεί γλώσσα LL1 γιατί δεν υπάρχει αναδρομή ( άμεση ή έμμεση ), δεν υπάρχουν κανόνες που να ξεκινούν με το ίδιο τερματικό σύμβολο ούτε και δυο που να παράγουν την κενή συμβολοσειρά, δηλαδή πληρούνται οι κανόνες της LL1. Φαίνεται και από τα σύνολα των PREDICT για τα οποία δεν υπάρχει κάποιο σύμβολο που να επαναλαμβάνεται σε predict άλλων κανόνων με το ίδιο σύμβολο στα αριστερά.

(β) Πίνακας συντακτικής ανάλυσης

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Κορυφή στοίβας | [ | ] | a | + | b | - | c |
| S | 1 | - | 2 | - | - | - | - |
| X | - | 3 | - | 4 | 5 | 5 | 3 |
| Y | - | 6 | - | - | 6 | 7 | 6 |

Περιγραφή εκτέλεσης

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Στοίβα** | **Είσοδος** | **Πράξη** |
| 0 | <S> | [a-acb] $ | Πρόβλεψη <S> ::== (<S>)<S> |
| 1 | ]<X><S>[ | [a-acb] $ | Ταίριασμα [ |
| 2 | ]<X><S> | a-acb] $ | Πρόβλεψη <S> ::== a |
| 3 | ]<X>a | a-acb] $ | Ταίριασμα a |
| 4 | ]<X> | -acb] $ | Πρόβλεψη <X> ::== <Y>b |
| 5 | ]b<Y> | -acb] $ | Πρόβλεψη <Y> ::== -<S><X>c |
| 6 | ]bc<X><S>- | -acb] $ | Ταίριασμα - |
| 7 | ]bc<X><S> | acb] $ | Πρόβλεψη <S> ::== a |
| 8 | ]bc<X>a | acb] $ | Ταίριασμα a |
| 9 | ]bc<X> | cb] $ | Πρόβλεψη <X> ::== є |
| 10 | ]bc | cb] $ | Ταίριασμα c |
| 11 | ]b | b] $ | Ταίριασμα b |
| 12 | ] | ] $ | Ταίριασμα ] |
| 13 | є | $ | **Αναγνώριση** |

Οι αναδρομικές κλήσεις ακολουθούν μια διάσχιση του δέντρου:  
  
****

**3**. Η σύγκρουση ολίσθησης-ελάττωσης είναι η σύγκρουση η οποία συμβαίνει στους συντακτικούς αναλυτές bottom-up κατά την οποία ο αναλυτής μπορεί και να ολισθήσει νέο χαρακτήρα στην στοίβα αλλά και να κάνει reduce κάποια περιεχόμενα σε κάποιο κανόνα. Επιλύεται χρησιμοποιώντας αναλυτές όπως LR(k), SLR(k) που υλοποιούν lookahead.  
Η παρακάτω γραμματική δεν είναι SLR(1) διότι οι παραγωγές <S>::=dc, <A>::=d έχουν σύγκρουση ελάττωσης-ελάττωσης.

# Ζήτημα 3

Για το ζήτημα 3 αντί για προεπεξεργαστή έχω χρησιμοποιήσει το πρόγραμμα σε bash com.sh στο οποίο χρησιμοποιώντας το εργαλέιο sed παράγω ένα αρχείο χωρίς comments για την χρησιμοποίηση με τον parser.  
Τα αρχεία περιέχονται στον φάκελο.

# Parser.l

%{   
#include <stdio.h>  
#include "parser.tab.h"  
int lineNum = 0;  
%}   
%option noyywrap

%%  
"program" return RW\_PROGRAM;  
"var" return RW\_VAR;  
"procedure" return RW\_PROCEDURE;  
"function" return RW\_FUNCTION;  
"if" return RW\_IF;  
"then" return RW\_THEN;  
"else" return RW\_ELSE;  
"while" return RW\_WHILE;  
"do" return RW\_DO;  
"begin" return RW\_BEGIN;  
"end" return RW\_END;  
"integer"|"boolean" return DATATYPE;  
":=" return ASSIGN;  
[0-9]+ return INT;  
"(" return '(';  
")" return ')';  
";" return ';';  
":" return ':';  
"," return ',';  
"." return '.';  
"=" return EQ;   
"<>" return NE;   
"<" return LT;   
">" return GT;   
">=" return GE;   
"<=" return LE;   
"\*" return TIMES;   
"div" return DIV;   
"mod" return MOD;   
"or" return OR;   
"and" return AND;   
"+" return PLUS;   
"-" return MINUS;   
"not" return NOT;  
[a-zA-Z][a-zA-Z0-9]\* return ID;  
"\n" {lineNum++;}  
"\t"|" " {}  
. return UNKNOWN\_TOKEN;  
%%

# parser.y

%{  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
extern FILE \*yyin;  
extern FILE \*yyout;  
void yyerror (char \*s);  
extern lineNum;  
%}

%token RW\_PROGRAM   
%token RW\_VAR   
%token RW\_PROCEDURE   
%token RW\_FUNCTION  
%token RW\_IF   
%token RW\_THEN   
%token RW\_ELSE  
%token RW\_WHILE   
%token RW\_DO   
%token RW\_BEGIN   
%token RW\_END  
%token ID  
%token DATATYPE  
%token ASSIGN  
%token INT  
%token UNKNOWN\_TOKEN  
%left OR  
%left AND  
%left NOT  
%left LT LE GT GE NE EQ  
%left PLUS MINUS  
%left TIMES MOD DIV   
%start program  
%%  
  
program:   
 RW\_PROGRAM ID ';' block '.'  
 ;  
block:   
 many\_locals compound\_statement  
 ;  
many\_locals:  
 |local\_definition many\_locals  
 ;

local\_definition:  
 variable\_definition   
 | procedure\_definition  
 | function\_definition  
 ;  
variable\_definition:   
 RW\_VAR def\_some\_variables ';' more\_vars  
 ;  
more\_vars:   
  
 | def\_some\_variables ';' more\_vars   
 ;   
def\_some\_variables:   
 ID more\_IDs ':' DATATYPE  
 ;  
more\_IDs:  
 | ',' ID more\_IDs  
 ;  
procedure\_definition: procedure\_header block ';'  
 ;  
procedure\_header:  
 RW\_PROCEDURE ID formal\_parameters ';'  
 ;  
function\_definition: function\_header block ';'  
 ;  
function\_header:  
 RW\_FUNCTION ID formal\_parameters ':' DATATYPE ';'  
 ;

formal\_parameters:  
 |'(' def\_some\_variables more\_parameters ')'  
 ;  
more\_parameters:  
 | ';' def\_some\_variables more\_parameters  
 ;  
statement:  
 | ID assignment\_or\_proc\_func\_call  
 | if\_statement  
 | while\_statement  
 | compound\_statement  
 ;  
assignment\_or\_proc\_func\_call:  
 ASSIGN expression  
 | actual\_parameters\_or\_not\  
 ;   
actual\_parameters\_or\_not:  
 |'(' actual\_parameters ')'  
 ;

if\_statement:  
 RW\_IF expression RW\_THEN statement else\_statement  
 ;  
else\_statement:  
 | RW\_ELSE statement  
 ;  
while\_statement:  
 RW\_WHILE expression RW\_DO statement  
 ;

actual\_parameters:  
 expression ',' actual\_parameters  
 | expression  
 ;  
compound\_statement:  
 RW\_BEGIN statement more\_statements RW\_END  
 ;  
more\_statements:  
 | ';' statement more\_statements  
 ;   
expression:  
 expression binary\_op expression  
 | unary\_op expression  
 | '(' expression ')'  
 | ID actual\_parameters\_or\_not  
 | INT  
 ;  
binary\_op:  
 PLUS   
 | MINUS  
 | TIMES  
 | DIV  
 | AND  
 | OR  
 | LE  
 | LT  
 | GE  
 | GT  
 | EQ  
 | NE  
 | MOD  
 ;  
unary\_op:  
 NOT  
 | PLUS  
 | MINUS   
 ;  
%%

void yyerror(char \*s){  
 printf("parser: Wrong Buzen program \n");  
 printf("parser: Error in line %d : %s \n", lineNum, s);  
}  
int main (int argc, char \*\*argv){  
 ++argv;   
 --argc;  
 int result;  
 char\* extension = 0;  
 if (argc > 0 ){  
 yyin = fopen(argv[0], "r");  
 if(yyin == 0){  
 printf("parser: %s: No such file  
\n", argv[0]);  
 fclose(yyin);  
 exit(1);  
 }  
 extension = strrchr(argv[0], '.');  
 if( !extension ||  
strcmp(extension,".buz") ){  
 printf("parser: %s: File format  
not recognized \n", argv[0]);  
 printf("parser: Target file  
should be .buz \n");  
 fclose(yyin);  
 exit(1);  
 }  
 yyout = fopen("output","w");  
 result = yyparse();  
 if(result == 0){  
 printf("parser: Corrent Buzen  
program \n");  
 }  
 fclose(yyin);  
 fclose(yyout);  
 return 0;  
 }else{  
 printf("parser: No input file \n");  
 printf("Usage: parser.exe targetfile.buz  
\n");  
 exit(1);  
 }  
}