CmpE 545 Term Project

An Artificial Neural Network Application On

Context Free Language Recognition

by
M. Toygar KARADENIZ

Bogazici University
1996

Introduction

Most of the programming language constructs have an inherently recursive structure that can be expressed by the Context-Free-Grammars (CFG). These type of grammars is easily understood and fully examined in several research areas. Many efficient parsing algorithms have been invented to recognize the sentences generated by those CFG's. The programming language compilers, mainly, parse the source code, also with doing some additional processes like symbol table maintenance or code generation.

Although, the main usage area of the CFG's currently, are the programming language structure definitions, some other language structures ,also, obey the rules governed by these type of grammars. One of the most important of these, is the 'Natural Language'. The structure of the Natural Language which we use everyday, is just suitable to be expressed as CFG. But the problem concerned with it, is that the recursive orderings of the terminals and of groups of terminals, has a great deal of importance in that case and recognition of whole sentences may be handled easily if the relative orderings taken into account in the first place rather than the absolute orderings in the full sentence. Artificial neural network application seems to be more suitable to handle the huge recognition phase of a natural sentence than a dedicated parser.

This project is about the implementation of an artificial neural network trained to determine the grammaticality of strings of syntactic categories in the Natural Language.

Description Of The Neural Net Used

In general, the design of a neural net should be matched to the intended problem area (although some nets such as the MLP have proved to be suitable for a wide range of low-level processing tasks). The network used in this project captures the order information in an input sentence by forming contiguous pairs, triples, 4-tuples etc. of pre-terminals from the sentence and storing each of these in an input node. In the broadest terms, it learns the tuples which are most commonly and strongly indicative of both grammatical and ungrammatical sentences, by exposure to a training set. This training set contains both 'yes' and 'no' examples and is generated by the program automatically from the given grammars. After training, the net should be able to generalize and make grammaticality judgments about unseen sentences from the same grammar. It works chiefly on short-range relationships (so no very long tuples are stored) and on neighboring symbols.

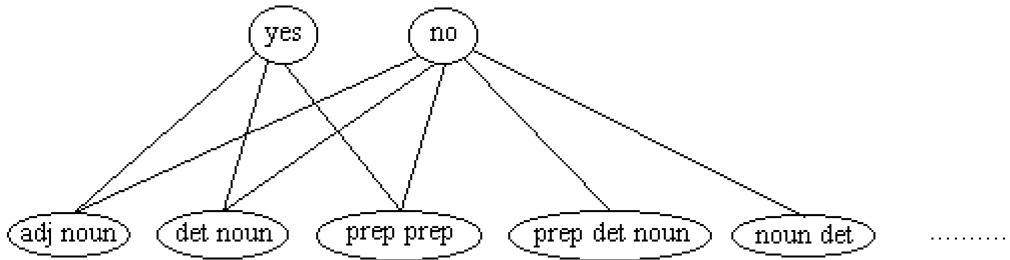


Figure 1

The architecture of the used neural network is simple. **Figure 1** shows a portion of it after a training. The number of the nodes in the input layer after training depends on the lengths of tuples selected and the number of syntactic categories in the grammar. This network is a single-layer net with output nodes calculating a simple weighted sum of their inputs using no threshold function. Only the output node with the highest activation fires. There is no hidden layer. The input layer is generally large and dynamically constructed. In this particular problem there are two output nodes and they are fixed. It is worth stating at this point why such an elementary architecture was chosen, when the MLP nets have proved so good. Although, it is true that back-propagation allows a hidden layer representation to be learned automatically, there are the advantages of speed of learning and convergence to a global minimum.

Weight Adjustment Strategy

Since the beginning of this project, several weight-adjustment strategies have been examined and tried, but none of them has proved better than the following one. Note that the weights are only adjusted when the network produces the wrong output. The links between the activated input nodes and the current output winning node (wrong output) are weakened, and the links between the activated input nodes and the desired response output node are strengthened. The weight adjustment function is given below :

$$W_{new} = (1+\Lambda * W_{old} / (1+(\Lambda * W_{old})^4)) * W_{old}$$

where Λ = + 1 for strengthening weights and - 1 for weakening weights.

Pseudo-Code Of The Training Algorithm

Order is the main concept in making decisions about the recognition of the grammar at hand. For example, ‘the ball is dropped’ is grammatically correct, but ‘dropped the is ball’ is incorrect. Our artificial neural network application will capture the order information in a sentence by making pairs, triples etc. from the terminals of the grammar and prepare a node for each of these orderings. For example, consider the following sentences : ‘the ball is dropped’ and ‘the small blue ball is dropped’. Suppose that our artificial neural network was trained on the first sentence. Then, it will be able to recognize the second sentence as well, although in most of the cases concerning CFG’s, addition of an extra terminal in front of the sentence will make it grammatically incorrect. In the most general sense, our artificial neural network will be able to learn the k-tuples that are the most strongly indicative of both grammatically correct and incorrect sentences by being trained over a training set. These k-tuples are also the most common ones. Then, it will be easy for it to generalize and recognize the unseen sentences from the same grammar.

A pseudo-code of our artificial neural network training algorithm is given below :

a) A sentence is encountered.
b) All orderings that constitute the k-tuples, are generated.
c) All matching k-tuples with the predetermined nodes in the net, is activated with a value of 1.
d) If there are no matching k-tuples with the new one, a new input node is created and the new one is dedicated to this new node.
e) The output node that has the largest value is determined.
f) The response-to-be-given is taken as an input from the training set. If it does not match the node determined in (e), it is added to the output layer. If they match, the weights between the node described in (e) and the input layer, are improved.

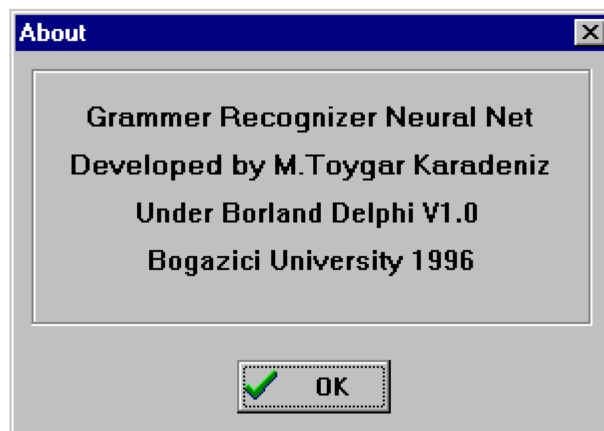
This algorithm is iterated through the training set to reach a certain level of stability.

Project Forms

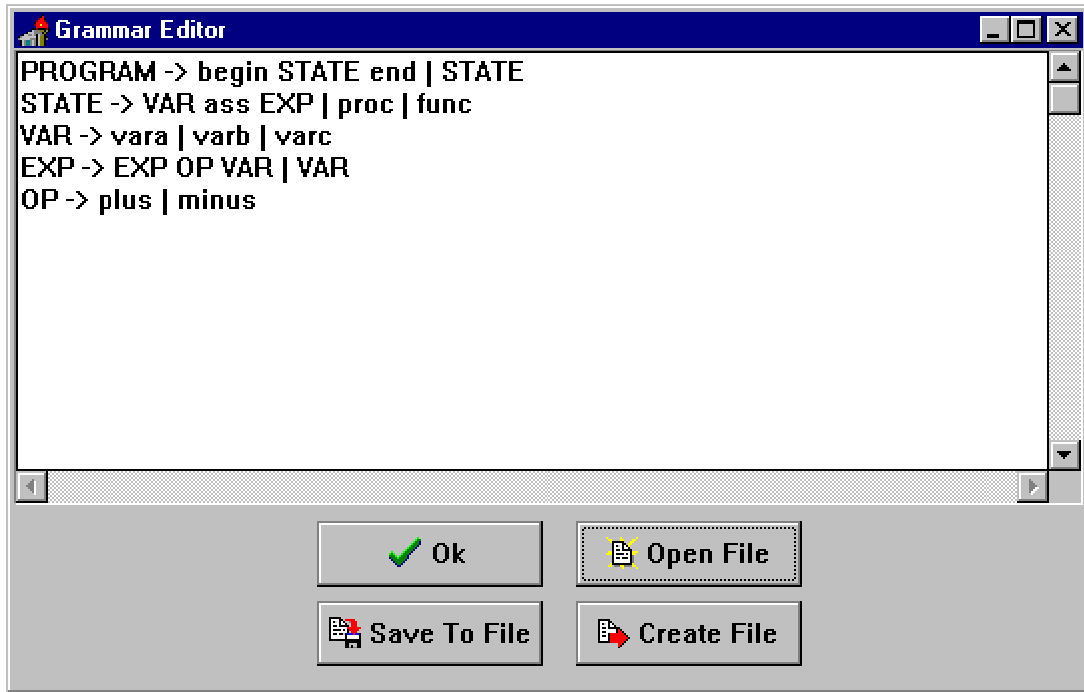
This project is implemented under Ms Windows using Borland Delphi V1.0. Following is the collection of the various windows that constructs the program's user interface.

The main menu of the program is given below. When the program starts it presents this menu to the user and waits for a response :

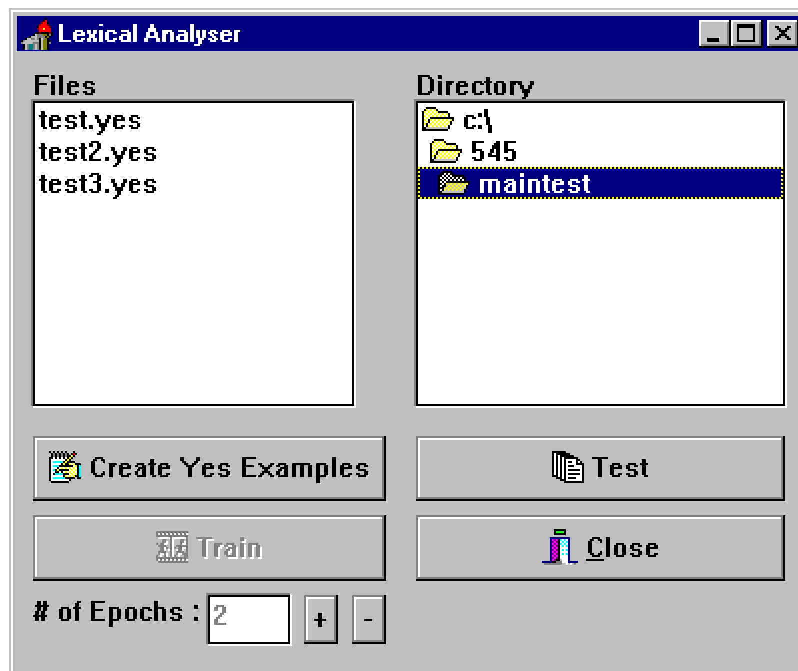
Pressing 'Quit' button exits the program, as obvious. 'About' button presents a window giving information about the program. This window is given below :



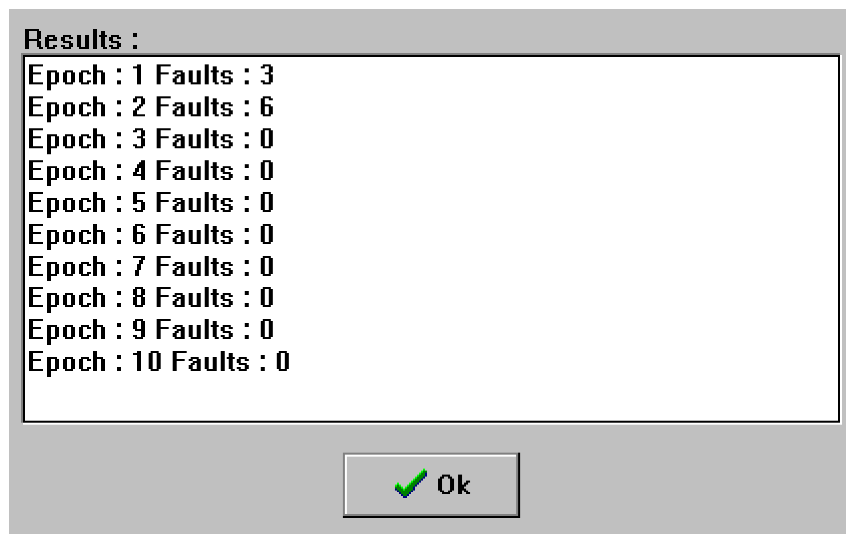
'Grammar Editor' button presents a classical editor window which is included in the project just for checking and writing the grammars to work with. It is given as the following for illustrative purposes :



'Neural CFC Checker' button takes the user to the main trainer and tester part of the network. The presented window has all the functionality concerning the project. This is also given below :



Training results are presented in a dedicated window which is given as the following :



Conclusion

In this project, a single layer, dynamically constructed neural net is implemented. This net has attacked problem of context free grammaticality determination and shown good results. There are mainly two ways in which this work may be extended. The first is to extend the formal grammar used to generate the input strings until it comes closer to natural English. Also, a decision is to be made as to what type of 'no' strings to use. The second extension may be to add a semantic analyzer into this net structure to not just examine the syntactic structures but the semantic meanings also.

References

[1] S. M. Lucas, R. I. Damper. *Syntactic Neural Networks*. Connection Science, Vol.2 , No. 3, 1990.

[2] E. Santos Jr. *A Massively Parallel Self-Tuning Context Free Parser*. Dept. of Computer Science, Brown University.

[3] Aho, Sethi and Ullman. Compilers : Principles, Techniques and Tools.

[4] M. R. Garey and D. S. Johnson. Computers and Intractability : A Guide to the theory of NP-Completeness. Freeman, San Francisco, (1979).

Appendix - Source Code Of The Program

```
unit About;

interface

uses WinTypes, WinProcs, Classes, Graphics, Forms, Controls,
StdCtrls, Buttons, ExtCtrls;

type
  TAboutBox = class(TForm)
    Panel1: TPanel;
    OKButton: TBitBtn;
    ProductName: TLabel;
    Copyright: TLabel;
    Label1: TLabel;
    Label2: TLabel;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  AboutBox: TAboutBox;

implementation

{$R *.DFM}

end.
```

```
unit Def_term;

interface

uses
  WinTypes, WinProcs, Classes, Graphics, Forms, Controls, StdCtrls,
  Buttons, ExtCtrls, SysUtils;

type
  TDefineVariable = class(TForm)
    Edit1: TEdit;
    BitBtn1: TBitBtn;
    Label1: TLabel;
    BitBtn2: TBitBtn;
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    Variable : string;
    Doit : boolean;
  end;

var
  DefineVariable: TDefineVariable;
```

```
implementation

{$R *.DFM}

procedure TDefineVariable.BitBtn1Click(Sender: TObject);
begin
 if Edit1.Text[1] in ['a'..'z','_']
 then begin
       Variable := Edit1.Text;
       Close;
      end
 else Application.MessageBox('First Letter Must Be In "a".."z","_"',
                             'Error', mb_OK);

end;

procedure TDefineVariable.BitBtn2Click(Sender: TObject);
begin
 Variable := '';
 Close;
end;

procedure TDefineVariable.Edit1Change(Sender: TObject);
begin
 if Edit1.Text = ''
 then BitBtn1.Enabled := False
 else BitBtn1.Enabled := True;
end;

procedure TDefineVariable.FormShow(Sender: TObject);
begin
 if not(doit) then Edit1.Text := ''
              else doit := false;
end;



procedure TDefineVariable.FormCreate(Sender: TObject);
begin
 Doit := false;
end;

end.
```

```
unit Editor;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
Controls, Forms, Dialogs, StdCtrls, Buttons, FCreate;


type
  TViewer = class(TForm)
    Memo1: TMemo;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    OpenDialog1: TOpenDialog;
    procedure FormResize(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
```

```
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    gram_name : string;
  end;

var
  Viewer: TViewer;

implementation

{$R *.DFM}

procedure TViewer.FormResize(Sender: TObject);
begin
     Memo1.Left := 1;
     Memo1.Top := 1;
     Memo1.Width := Viewer.Width - 10;
     Memo1.Height := Viewer.Height - 120;
     BitBtn1.Left := (Viewer.Width DIV 2) - 120;
     BitBtn2.Left := (Viewer.Width DIV 2) - 120;
     BitBtn3.Left := (Viewer.Width DIV 2) + 10;
     BitBtn4.Left := (Viewer.Width DIV 2) + 10;
     BitBtn1.Top := Viewer.Height - 110;
     BitBtn2.Top := Viewer.Height - 70;
     BitBtn3.Top := Viewer.Height - 110;
     BitBtn4.Top := Viewer.Height - 70;
end;

procedure TViewer.BitBtn1Click(Sender: TObject);
begin
 Close
end;

procedure TViewer.BitBtn2Click(Sender: TObject);
begin
 Memo1.Lines.SaveToFile(gram_name);
end;



procedure TViewer.FormShow(Sender: TObject);
begin
 Viewer.Memo1.Lines.Clear;
end;

procedure TViewer.BitBtn3Click(Sender: TObject);
var F : file of byte;
begin
  if OpenDialog1.Execute then begin
    AssignFile(F, OpenDialog1.FileName);
    {$I-}
    Reset(F);
    {$I+}
    if IOResult = 0 then
    begin
     gram_name := OpenDialog1.FileName;
     Memo1.Lines.Clear;
     Memo1.Lines.LoadFromFile(gram_name);
    end
    else
      MessageDlg('File Access Error', mtWarning, [mbOk], 0);
  end;
```

```
end;

procedure TViewer.BitBtn4Click(Sender: TObject);
begin
 Create_File.Namex := '';
 Create_File.ShowModal;
 Gram_Name := Create_File.Namex;
 Memo1.Lines.Clear;
end;

end.
```

```
unit Fcreate;

interface

uses WinTypes, WinProcs, Classes, Graphics, Forms, Controls,
StdCtrls, Buttons, Dialogs;

type
  TCreate_File = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    OKBtn: TBitBtn;
    CancelBtn: TBitBtn;
    procedure Edit1Change(Sender: TObject);
    procedure OKBtnClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    Namex : string;
  end;

var
  Create_File: TCreate_File;

implementation

{$R *.DFM}

procedure TCreate_File.Edit1Change(Sender: TObject);
begin
 if Edit1.Text = '' then OkBtn.Enabled := False
                    else OkBtn.Enabled := True;
end;

procedure TCreate_File.OKBtnClick(Sender: TObject);
var F : file of byte;
begin
 AssignFile(F, Edit1.Text);
 {$I-}
 Reset(F);
 {$I+}
 if IOResult = 0
 then
  MessageDlg('File Already Present', mtWarning, [mbOk], 0)
 else
 begin
  Rewrite(F);
  Namex := Edit1.Text;
 end
end;

end.
```

```
unit Fprocess;

interface

uses WinTypes, WinProcs, Classes, Graphics, Forms, Controls,
StdCtrls,
  Buttons, ExtCtrls, SysUtils, Lex_Res;

type
  TFile_Process = class(TForm)
    Timer1: TTimer;
    Panel1: TPanel;
    procedure FormShow(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    FileToCheck : string;
    Yes, error_flag : boolean;
  end;

var
  File_Process: TFile_Process;
  infile : TextFile;
  lex_arr : array [1..50,1..50] of string[10];
  mainch : char;
  upper, arrow : boolean;
  lex_y : integer;
  lex_x : array [1..50] of integer;
  lex_alt : array [1..50] of integer;

implementation

{$R *.DFM}

function Handle_Blanks(var thefile : Textfile; var end_of_file :
boolean) : char;
var ch : char;
begin
 end_of_file := False;
 read(infile,ch);
 if eof(infile) then end_of_file := True;
 while ((ch=' ') or (ord(ch)=13) or (ord(ch)=10))
       and not(eof(infile)) do
 begin
  if (ord(ch)=13) or (ord(ch)=10) then
  begin
   arrow := False;
   upper := False;
   lex_y := lex_y + 1;
  end;
  read(infile,ch);
 end;
 Handle_Blanks := ch;
end;

procedure Lex_Error;
begin
 Lex_Result.Label1.Caption := 'Lexical Error In Grammar ...';
 Lex_Result.ShowModal;
 File_Process.error_flag := True;
end;
```

```
function Lexer : string;
label out;
var i : integer;
    ch : char;
    collect_str : string;
    end_of_file, first : boolean;
begin
 if mainch = '0' then
 begin
  ch := Handle_Blanks(infile,end_of_file);
  mainch := ch;
 end;
 collect_str := '';
 if not(end_of_file) then
 begin
  first := True;
  while (IsCharUpper(ch)) do
  begin
   if first then
   begin
    if upper and not(arrow) then
    begin
     Lex_Error;
     exit
    end;
    upper := True;
    first := False;
   end;
   collect_str := collect_str + ch;
   read(infile,ch);
   mainch := '0';
  end;
  if collect_str <> '' then goto out;
  if (collect_str = '') and (ch = '-') then
  begin
   read(infile,ch);
   if ch<>'>' then begin
                    Lex_Error;
                    exit
                   end
              else collect_str := '->';
   if arrow or not(upper) then
   begin
    Lex_Error;
    exit
   end;
   arrow := True;
   upper := False;
   mainch := '0';
  end;
  if collect_str <> '' then goto out;
  if (collect_str = '') and (ch = '|') then
  begin
   collect_str := '|';
   mainch := '0';
  end;
  if collect_str <> '' then goto out;
  while (IsCharLower(ch)) do
  begin
   if not(arrow) then
   begin
    Lex_Error;
    exit
   end;
   collect_str := collect_str + ch;
   read(infile,ch);
```

```
   mainch := '0';
  end;
 end;
 if ((collect_str = '') or ((ch <> ' ')
    and (ord(ch)<>13) and (ord(ch)<>10)))
    and not(end_of_file)
    then begin
          Lex_Error;
          exit
         end;
 if (end_of_file) then collect_str := '0end';
out :
 Lexer := collect_str;
end;

procedure Generate_Ex;forward;

procedure TFile_Process.FormShow(Sender: TObject);
var i : integer;
begin
 mainch := '0';
 for i :=1 to 50 do
  lex_x[i] := 0;
 for i :=1 to 50 do
  lex_alt[i] := 1;
 lex_y := 1;
 AssignFile(infile,FileToCheck);
 Reset(infile);
 Timer1.Enabled := True;
end;

procedure TFile_Process.Timer1Timer(Sender: TObject);
var cur_str : string;
    i,j : integer;
begin
 Timer1.Enabled := False;
 arrow := False;
 upper := False;
 error_flag := False;
 cur_str := Lexer;
 if error_flag then exit;
 lex_x[lex_y] := lex_x[lex_y] + 1;
 lex_arr[lex_y,lex_x[lex_y]] := cur_str;
 if cur_str = '|' then lex_alt[lex_y] := lex_alt[lex_y] + 1;
 while (cur_str <> '0end') do
 begin
  error_flag := False;
  cur_str := Lexer;
  if error_flag then break;
  if (cur_str <> '0end') then
  begin
   lex_x[lex_y] := lex_x[lex_y] + 1;
   lex_arr[lex_y,lex_x[lex_y]] := cur_str;
   if cur_str = '|' then lex_alt[lex_y] := lex_alt[lex_y] + 1;
  end
 end;
 if not(error_flag) then
 begin
   Lex_Result.Label1.Caption := 'No Errors Found In Grammar ...';
   Lex_Result.ShowModal;
{  writeln;
  for i:=1 to lex_y do
  begin
   for j:=1 to lex_x[i] do
    write(lex_arr[i,j],' ');
   write(lex_alt[i]);
```

```
   writeln;
  end;}
  Generate_Ex;
 end;
 CloseFile(infile);
 Close;
end;


procedure Generate_Ex;
const max_written = 50;
type myrec = record
             str : string;
            end;
var c,i,j,l,written : integer;
    outfile : text;
    generate : array [1..500] of string[10];
    iteration,where,startpl,endpl,k,accum,gen_ind,count : integer;
    which : real;
    found,entered,all_ok : boolean;
    tempfile : file of myrec;
    a_str : string;
    arec : myrec;
label start_over;
begin
 written := 0;
 Randomize;
 for i:=length(File_Process.FileToCheck)-2 to
length(File_Process.FileToCheck) do
  if File_Process.Yes then
  begin
   if i=length(File_Process.FileToCheck)-2 then
File_Process.FileToCheck[i] := 'e';
   if i=length(File_Process.FileToCheck)-1 then
File_Process.FileToCheck[i] := '_';
   if i=length(File_Process.FileToCheck) then
File_Process.FileToCheck[i] := 'y';
  end
  else
  begin
   if i=length(File_Process.FileToCheck)-2 then
File_Process.FileToCheck[i] := 'e';
   if i=length(File_Process.FileToCheck)-1 then
File_Process.FileToCheck[i] := '_';
   if i=length(File_Process.FileToCheck) then
File_Process.FileToCheck[i] := 'n';
  end;
 AssignFile(outfile,File_Process.FileToCheck);
 AssignFile(tempfile,'_temp_.tmp');
 Rewrite(tempfile);
 Rewrite(outfile);
 iteration := 0;
 while (written < max_written) and (iteration < 1000) do
 begin
 start_over :
  iteration := iteration + 1;
  gen_ind := 1;
  generate[gen_ind] := lex_arr[1,1];
  all_ok := False;
  while not(all_ok) do
  begin
   entered := False;
   for i := 1 to gen_ind do
   begin
    if IsCharUpper(generate[i,1]) then
    begin
```

```
     entered := True;
     for j := 1 to lex_y do
      if (generate[i] = lex_arr[j,1]) then
      begin
       which := Random;
       where := Round(which*lex_alt[j]);
       if where = 0 then where := 1;
       accum := 1;
       for k:=2 to lex_x[j] do
        if accum = where
        then begin
              if where = 1 then startpl := 3
                           else startpl := k;
              for l := startpl to lex_x[j] do
               if l = lex_x[j]
               then endpl := lex_x[j]
               else if lex_arr[j,l] = '|'
                    then begin
                          endpl := l - 1;
                          break
                         end;
              break;
             end
        else if lex_arr[j,k] = '|' then accum := accum + 1;
       for k := gen_ind downto i+1 do
        generate[k+endpl-startpl] := generate[k];
       for k := i to i+endpl-startpl do
       begin
        generate[k] := lex_arr[j,startpl+k-i];
        if k<>i then gen_ind := gen_ind + 1;
        if gen_ind > 200 then goto start_over;
       end;
       break
      end
    end
   end;
   if not(entered) then all_ok := True;
  end;
  a_str := '';
  for c := 1 to gen_ind do
   a_str := a_str + generate[c];
  found := false;
  Reset(tempfile);
  while not(found) and not(eof(tempfile)) do
  begin
   read(tempfile,arec);
   if arec.str = a_str then found := true;
  end;
  if not(found) then
  begin
   arec.str := a_str;
   seek(tempfile,filesize(tempfile));
   write(tempfile,arec);
   CloseFile(tempfile);
   for c := 1 to gen_ind do
    writeln(outfile,generate[c]);
   writeln(outfile,'>');
   if File_Process.Yes then writeln(outfile,'"Yes')
                       else writeln(outfile,'"No');
   written := written + 1;
{   writeln(written);}
  end
 end;
 CloseFile(outfile);
 DeleteFile('_temp_.tmp');
end;
```

```
end.
```

```
unit Lex_res;

interface

uses WinTypes, WinProcs, Classes, Graphics, Forms, Controls,
StdCtrls,
  Buttons, ExtCtrls;

type
  TLex_Result = class(TForm)
    Panel1: TPanel;
    OKButton: TBitBtn;
    Label1: TLabel;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Lex_Result: TLex_Result;

implementation

{$R *.DFM}

end.
```

```
unit Lexer;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
Controls,
  Forms, Dialogs, FileCtrl, StdCtrls, Buttons, Fprocess, ExtCtrls,
WinCrt,
  Res, Test_Str;

const MIN = 2;
      MAX = 3;
      max_node_num = 150;
type
  TLexical_Analyser = class(TForm)
    FileListBox1: TFileListBox;
    DirectoryListBox1: TDirectoryListBox;
    BitBtn1: TBitBtn;
    Label1: TLabel;
    Label2: TLabel;
    Timer1: TTimer;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    Edit1: TEdit;
    Label3: TLabel;
    BitBtn4: TBitBtn;
    BitBtn5: TBitBtn;
    BitBtn6: TBitBtn;
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
```

```
    procedure FormShow(Sender: TObject);
    procedure DirectoryListBox1Change(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  string10 = string[10];
  string30 = string[30];

var
  Lexical_Analyser: TLexical_Analyser;
  nodes : array [0..max_node_num] of string30;
  node_cnt : integer;
  node_adj : array [0..max_node_num,0..1] of boolean;
  node_weights : array [0..max_node_num,0..1] of real;
  yes_file,no_file : string;

implementation

{$R *.DFM}

procedure TLexical_Analyser.BitBtn1Click(Sender: TObject);
begin
 Close
end;


procedure mix_two_files(ff,sf,tf : string);forward;

procedure TLexical_Analyser.BitBtn2Click(Sender: TObject);
var i : integer;
begin
 File_Process.FileToCheck := FileListBox1.FileName;
 if File_Process.FileToCheck <> '' then
 begin
  if BitBtn2.Caption = 'Create Yes Examples' then File_Process.Yes :=
True;
  if BitBtn2.Caption = 'Create No Examples' then File_Process.Yes :=
False;
  File_Process.ShowModal;
  if not(File_Process.error_flag) then
   if BitBtn2.Caption = 'Create Yes Examples' then
   begin
    yes_file := File_Process.FileToCheck;
    FileListBox1.Mask := '*.no_';
    FileListBox1.Refresh;
    BitBtn2.Caption := 'Create No Examples';
   end
   else
   begin
    no_file := File_Process.FileToCheck;
    BitBtn2.Caption := 'Create Yes Examples';
    BitBtn2.Enabled := False;
    BitBtn3.Enabled := True;
    for i:=length(File_Process.FileToCheck)-2 to
length(File_Process.FileToCheck) do
    begin
```

```
     if i=length(File_Process.FileToCheck)-2 then
File_Process.FileToCheck[i] := 'e';
     if i=length(File_Process.FileToCheck)-1 then
File_Process.FileToCheck[i] := 'p';
     if i=length(File_Process.FileToCheck) then
File_Process.FileToCheck[i] := 'l';
    end;
    mix_two_files(yes_file,no_file,File_Process.FileToCheck);
    FileListBox1.Mask := '*.epl';
    FileListBox1.Refresh;
   end;
 end
end;

procedure TLexical_Analyser.FormShow(Sender: TObject);
begin
 FileListBox1.Mask := '*.yes';
 FileListBox1.Refresh;
 FileListBox1.Directory := DirectoryListBox1.Directory;
 FileListBox1.Update;
end;

procedure TLexical_Analyser.DirectoryListBox1Change(Sender: TObject);
begin
 Timer1.Enabled := True;
end;

procedure TLexical_Analyser.Timer1Timer(Sender: TObject);
begin
 Timer1.Enabled := False;
 FileListBox1.Directory := DirectoryListBox1.Directory;
 FileListBox1.Update;
end;

procedure get_tuples(mystr : array of string10;mycnt : integer;
                     var generated : array of string30; var gen_ind :
integer);
var cnt,i,j,k : integer;
    to_gen : string;
    found : boolean;
begin
 gen_ind :=0;
 for cnt := MIN to MAX do
 begin
  for i := 0 to mycnt-cnt do
  begin
   generated[gen_ind] := '';
   to_gen := '';
   for j := 0 to cnt-1 do
    to_gen := to_gen + ' ' + mystr[i+j];
   found := false;
   for k := 0 to gen_ind - 1 do
    if generated[k] = to_gen then
    begin
     found := true;
     break
    end;
   if not(found) then
   begin
    generated[gen_ind] := to_gen;
    gen_ind := gen_ind + 1;
   end
  end
 end;
end;
```

```
procedure read_strings(var mystr : array of string10; var mycnt :
integer;
                       var myopin : string10; var myfile : textfile);
var read_str : string10;
begin
 mycnt := 0;
 readln(myfile,read_str);
 while (read_str <> '>') do
 begin
  mystr[mycnt] := read_str;
  mycnt := mycnt + 1;
  readln(myfile,read_str);
 end;
 readln(myfile,myopin);
end;

function is_present(astr : string30; var where : integer) : boolean;
var i : integer;
begin
 result := false;
 where := 0;
 for i := 0 to node_cnt-1 do
  if astr = nodes[i] then
  begin
   result := true;
   where := i;
   break
  end
end;



procedure mix_two_files(ff,sf,tf : string);
var one_str : string;
    file1,file2,file3 : textfile;
begin
 assignfile(file1,ff);
 assignfile(file2,sf);
 assignfile(file3,tf);
 reset(file1);
 reset(file2);
 rewrite(file3);
 while not(eof(file1)) and not(eof(file2)) do
 begin

  readln(file1,one_str);
  while (one_str <> '"Yes') and (one_str <> '"No') do
  begin
   writeln(file3,one_str);
   readln(file1,one_str);
  end;
  writeln(file3,one_str);

  readln(file2,one_str);
  while (one_str <> '"Yes') and (one_str <> '"No') do
  begin
   writeln(file3,one_str);
   readln(file2,one_str);
  end;
  writeln(file3,one_str);

 end;

 if eof(file1)
 then while not(eof(file2)) do
      begin
```

```
       readln(file2,one_str);
       writeln(file3,one_str);
      end
 else while not(eof(file1)) do
      begin
       readln(file1,one_str);
       writeln(file3,one_str);
      end;

 closefile(file1);
 closefile(file2);
 closefile(file3);

 DeleteFile(ff);
 DeleteFile(sf);

end;


function new_weight(old_weight : real; weaken : boolean) : real;
var delta : integer;
begin
 if weaken then delta := -1
           else delta := 1;
 result := (1 +
((delta*old_weight)/(1+sqr(sqr(delta*old_weight)))))*old_weight;
 new_weight := result;
end;

procedure TLexical_Analyser.BitBtn3Click(Sender: TObject);
var cur_epoch,max_epoch : integer;
    myfile : textfile;
    mycnt,gen_ind,i,where,num_of_faults : integer;
    myopin : string10;
    mystr : array [0..max_node_num] of string10;
    generated : array [0..max_node_num] of string30;
    yes_val,no_val : real;
    st : string;
begin
 Results.Memo1.Lines.Clear;
 if Lexical_Analyser.FileListBox1.FileName <> '' then
 begin
  randomize;
  for i := 0 to max_node_num do
  begin
   node_adj[i,0] := false;
   node_adj[i,1] := false;
   node_weights[i,0] := random;
   node_weights[i,1] := random;
  end;
  max_epoch := StrToInt(Edit1.Text);
  for cur_epoch := 1 to max_epoch do
  begin
   num_of_faults := 0;
   AssignFile(myfile,Lexical_Analyser.FileListBox1.FileName);
   Reset(myfile);
   while not(eof(myfile)) do
   begin
    read_strings(mystr,mycnt,myopin,myfile);
    get_tuples(mystr,mycnt,generated,gen_ind);

    if node_cnt = 0 then              { do it only once - just
connect }
    begin
     for i := 0 to gen_ind - 1 do
     begin
```

```
      if not(is_present(generated[i],where)) then
      begin
       nodes[node_cnt] := generated[i];
       node_adj[node_cnt,0] := true;
       node_adj[node_cnt,1] := true;
       node_cnt := node_cnt + 1;
      end
     end;
    end
    else
    begin
     yes_val := 0;
     no_val := 0;
     for i := 0 to gen_ind - 1 do
      if is_present(generated[i],where) then    { activation function
}
      begin
       if node_adj[where,0] then yes_val := yes_val +
node_weights[where,0];
       if node_adj[where,1] then no_val := no_val +
node_weights[where,1]
      end;

     if (myopin='"Yes') and (yes_val < no_val) then
     begin
      for i := 0 to gen_ind - 1 do
       if is_present(generated[i],where) then
       begin
        node_weights[where,0] :=
new_weight(node_weights[where,0],false);
        node_weights[where,1] :=
new_weight(node_weights[where,1],true);
       end;
      num_of_faults := num_of_faults + 1;
     end
     else if (myopin='"No') and (yes_val >= no_val) then
     begin
      for i := 0 to gen_ind - 1 do
       if is_present(generated[i],where) then
       begin
        node_weights[where,0] :=
new_weight(node_weights[where,0],true);
        node_weights[where,1] :=
new_weight(node_weights[where,1],false);
       end;
      num_of_faults := num_of_faults + 1;
     end;

     for i := 0 to gen_ind - 1 do
      if not(is_present(generated[i],where)) then
      begin
       nodes[node_cnt] := generated[i];
       node_cnt := node_cnt + 1;
      end;


     if myopin = '"Yes' then
      for i := 0 to node_cnt - 1 do
       node_adj[i,0] := true
     else
      for i := 0 to node_cnt - 1 do
       node_adj[i,1] := true;


    end
   end;
```

```
   CloseFile(myfile);
   st := 'Epoch : ' + IntToStr(cur_epoch) + ' Faults : ' +
IntToStr(num_of_faults);
   Results.Memo1.Lines.Add(st);
  end;
  Results.ShowModal;
 end
end;

procedure TLexical_Analyser.Edit1Change(Sender: TObject);
begin
 if Edit1.Text[1] in ['0'..'9']
    then
    else Edit1.Text := '';
end;

procedure TLexical_Analyser.BitBtn4Click(Sender: TObject);
begin
 Edit1.Text := IntToStr(StrToInt(Edit1.Text)+1);
end;

procedure TLexical_Analyser.BitBtn5Click(Sender: TObject);
begin
 if StrToInt(Edit1.Text) > 1
 then Edit1.Text := IntToStr(StrToInt(Edit1.Text)-1);
end;

procedure TLexical_Analyser.BitBtn6Click(Sender: TObject);
var mystr : array [0..150] of string10;
    mycnt : integer;
    i, gen_ind, where : integer;
    generated : array [0..150] of string30;
    yes_val, no_val : real;
begin
 WriteLine.ShowModal;
 if WriteLine.WL_Num <> 0 then
 begin
  for i := 0 to 150 do
  begin
   mystr[i] := WriteLine.WL[i];
   mycnt := WriteLine.WL_Num;
  end;
  get_tuples(mystr,mycnt,generated,gen_ind);
  yes_val := 0;
  no_val := 0;
  for i := 0 to gen_ind - 1 do
   if is_present(generated[i],where) then    { activation function }
   begin
    if node_adj[where,0] then yes_val := yes_val +
node_weights[where,0];
    if node_adj[where,1] then no_val := no_val +
node_weights[where,1]
   end;
  Results.Memo1.Lines.Clear;
  if yes_val > no_val
     then Results.Memo1.Lines.Add('Yes')
     else if yes_val < no_val
          then Results.Memo1.Lines.Add('No')
          else Results.Memo1.Lines.Add('Not Determined');
  Results.ShowModal;
 end
end;

end.
```

```
unit Main;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
Controls,
  Forms, Dialogs, StdCtrls, Buttons, Lexer, Editor, About, ExtCtrls;

type
  TMain_Menu = class(TForm)
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    Grammar_Editor: TBitBtn;
    Bevel1: TBevel;
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure Grammar_EditorClick(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Main_Menu: TMain_Menu;

implementation

{$R *.DFM}

procedure TMain_Menu.BitBtn1Click(Sender: TObject);
begin
 Lexical_Analyser.FileListBox1.Mask := '*.*';
 Lexical_Analyser.BitBtn2.Enabled := True;
 Lexical_Analyser.BitBtn3.Enabled := False;
 Lexical_Analyser.Show;
end;

procedure TMain_Menu.BitBtn2Click(Sender: TObject);
begin
 Halt
end;

procedure TMain_Menu.Grammar_EditorClick(Sender: TObject);
var  F: file of Byte;
begin
 Viewer.Gram_Name := '';
 Viewer.Show;
end;

procedure TMain_Menu.BitBtn3Click(Sender: TObject);
begin
 AboutBox.ShowModal;
end;

end.
```

```
unit Res;

interface
```

```
uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics,
Controls,
  Forms, Dialogs, ExtCtrls, StdCtrls, Buttons;

type
  TResults = class(TForm)
    Memo1: TMemo;
    BitBtn1: TBitBtn;
    Bevel1: TBevel;
    Label1: TLabel;
    procedure BitBtn1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Results: TResults;

implementation

{$R *.DFM}

procedure TResults.BitBtn1Click(Sender: TObject);
begin
 Close
end;

end.
```

```
unit Test_str;

interface

uses
  WinTypes, WinProcs, Classes, Graphics, Forms, Controls, StdCtrls,
  Buttons, ExtCtrls, SysUtils, Def_Term;

type
  TWriteLine = class(TForm)
    ListBox1: TListBox;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure FormShow(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    WL : array [1..150] of string[10];
    WL_Num : integer;
  end;

var
  WriteLine: TWriteLine;
```

```
implementation

{$R *.DFM}

procedure TWriteLine.BitBtn4Click(Sender: TObject);
begin
 WL_Num := 0;
 Close;
end;

procedure TWriteLine.BitBtn3Click(Sender: TObject);
var i : integer;
begin
 for i := 0 to ListBox1.Items.Count-1 do
  WL[i] := ListBox1.Items[i];
 WL_Num := ListBox1.Items.Count;
 Close;
end;

procedure TWriteLine.BitBtn1Click(Sender: TObject);
begin
 DefineVariable.Caption := 'Add A Terminal';
 DefineVariable.ShowModal;
 if DefineVariable.Variable <> '' then
 ListBox1.Items.Add(DefineVariable.Variable);
end;

procedure TWriteLine.BitBtn2Click(Sender: TObject);
var i : integer;
begin
 for i:=0 to ListBox1.Items.Count-1 do
  if ListBox1.Selected[i] then
  begin
   ListBox1.Items.Delete(i);
   break;
  end
end;

procedure TWriteLine.Timer1Timer(Sender: TObject);
begin
 if ListBox1.Items.Count = 0
    then begin
          BitBtn2.Enabled := False;
          BitBtn3.Enabled := False
         end
    else begin
          BitBtn2.Enabled := True;
          BitBtn3.Enabled := True;
         end
end;

procedure TWriteLine.FormShow(Sender: TObject);
begin
 ListBox1.Items.Clear;
end;

end.
```