An Delphi Application for the Syntactic and Lexical Analysis of a Phrase Using Cocke, Younger, and Kasami Algorithm

Bogdan Pătruț
Department of Mathematics and Computer Science,
Faculty of Sciences, “Vasile Alecsandri” University of Bacău
Calea Mărășești, 157, 600115, Bacău, Romania
bogdan@edusoft.ro

Ioana Boghian
Department of Philosophy and Communication Sciences,
Faculty of Letters, “Vasile Alecsandri” University of Bacău
Spiru Haret, 8, 600114, Bacău, Romania
rahela_bac@yahoo.com.uk

Abstract
This paper focus on the Cocke, Younger, and Kasami algorithm. We present a Delphi application that analyzes the lexicon and the syntax of a sentence in Romanian. We use a Chomsky normal form (CNF) grammar. We will present the source of a Delphi implementation of the CKY algorithm.

Keywords: CKY algorithm, lexical analysis, Delphi programming

1. General presentation
The program, which is highly complex, is part of the category of natural language processing programs. A phrase is entered into a text box and the program tells whether the phrase is correct or not, according to some vocabulary and an array of previously established syntactic rules which make up a certain grammar [1].

![Basic CKY Parser](image)

Figure 1. Example of parsing the phrase “orice barbat iubeste o femeie frumoasa si desteapta” (“any man loves a beautiful and intelligent woman”)

For example, in the figure above (Figure 1), the phrase “orice barbat iubeste o femeie frumoasa si desteapta” (“any man loves a beautiful and intelligent woman”) has been entered and has been accepted as correct. The table shows the diagram of the syntactic analysis for the phrase,
which starts from the row labelled with “0” and ends with the column labelled “desteapta” (“intelligent”).

The lexicon used in the analysis, written in the 'LEX.TXT' text file, was the following:

Det->orice  
Det->fiecare  
Det->o  
Det->un  
Pron->el  
N->barbat  
N->femeie  
V->iubeste  
V->uraste  
A->frumoasa  
A->desteapta  
C->si  
C->sau  

Thus, the previously analysed phrase contains words only from the chosen vocabulary. On the other hand, the phrase "any dog hates a cat", although syntactically correct in Romanian, is not accepted because it contains words that have not been entered into our vocabulary.

The file contains, on each row, a rule of the type (2):

\[ \text{GM} \rightarrow \text{W} \quad \text{(2)} \]

where GM is a grammatical category (or part of speech) and W is an word from a certain dictionary.

Syntax rules (from the 'GRAM.TXT' file) constitute a subset of the Romanian syntax rules:

\[ \text{S} \rightarrow \text{NP} \ \text{VP} \]
\[ \text{NP} \rightarrow \text{Pron} \]
\[ \text{NP} \rightarrow \text{N} \]
\[ \text{NP} \rightarrow \text{Det N} \]
\[ \text{NP} \rightarrow \text{NP AP} \]
\[ \text{AP} \rightarrow \text{A} \]
\[ \text{AP} \rightarrow \text{AP CP} \]
\[ \text{CA} \rightarrow \text{C A} \]
\[ \text{VP} \rightarrow \text{V VP} \]
\[ \text{VP} \rightarrow \text{V NP} \]

Thus, by using the established notation (S = sentence, NP = noun phrase, VP = verb phrase, N = noun, Det = determiner (article), AP = adjectival phrase, A = adjective, C = conjunction, V = verb, CA = group made up of a conjunction and an adjective.

The syntax rules are given, one on each line, thus (3):

\[ \text{GM1} \rightarrow \text{GM2 GM3} \quad \text{(3)} \]

meaning that the first grammatical category (GM1) forms out of the concatenation of the other two (GM2 and GM3), from the right side of the arrow.

According to the syntax rules used in our application, the phrase we have analysed in the beginning is correct, while the following figure (Figure 2) presents a case viewed as incorrect.
We should mention the fact that the algorithm that we have implemented uses only grammars written in Chomsky (CNF) normal format. A CNF grammar is a context-free grammar where productions take the form of: \( A \rightarrow B \, C \), where \( A \) is a non-terminal, and \( B \) and \( C \) are non-terminals or pre-terminals [1], [5]. We have also considered rules of the type \( A \rightarrow B \) as acceptable. Therefore, in the right side of the production rules there will be two or only one element.

Details concerning the topic under discussion, for the unknowing reader, can be studied in the works mentioned as references.

### 2. The Cocke, Younger, and Kasami Algorithm

We will further present the basic analysis algorithm (also called "chart-parsing") elaborated by Cocke, Younger, and Kasami and named the basic CYK algorithm [2], [5].

The algorithm uses a matrix (a diagram) chart, like those in the previous figures.

First of all, we need certain definitions:

The following operation is defined (4):

\[
\text{Star}(X,Y) = \{C | (A \in X) \land (B \in Y) \land (C \rightarrow AB \in \text{Rules}) \}
\]

where \( \text{Rules} \) denotes the production rules of the grammar.

This represents the fact that the product of two cells from the matrix is created by combining all the pairs of items in the two cells which satisfy certain grammar rules.

Another operation that is defined is (5):

\[
\text{Closure}(S) = \{A | (A \in S) \lor ((B \in \text{Closure}(S)) \land (A \rightarrow B \in \text{Rules}) \}
\]

This represents the fact that closing an \( S \) cell is formed of the content of \( S \) plus the result of adding any other category deriving from an existing member of \( S \)'s closure. For example, if \( N \) is in \( S \) then \( N \) belongs to \( \text{Closure}(S) \); then, if there is a rule \( NP \rightarrow N \), and \( NP \) will be added to \( \text{Closure}(S) \); things continue thus as long as new members can still be added to \( \text{Closure}(S) \).

Finally, there is also a Lookup function, of the type (6):

\[
\text{Lookup}(k) = \{A | A \rightarrow \text{word}_k\}
\]
meaning that it gives us the list of grammatical categories that the word number \( k \) from our phrase provides (sometimes a word can have more than one grammatical category, for example “going” can be both a noun or a verb).

Taking into consideration the above definitions, the basic CKY algorithm is [4]:

$$\begin{align*}
\text{for } & k := 1 \text{ to } n \text{ do} \\
\text{begin} & \\
& \text{chart}[k-1,k] := \text{Closure} (\text{Lookup}(k)); \\
& \text{for } i := k-2 \text{ downto } 0 \text{ do} \\
& \text{begin} \\
& \text{chart}[i,k] := \emptyset; \\
& \text{for } j := k-1 \text{ downto } i+1 \text{ do} \\
& \text{chart}[i,k] := \text{chart}[i,k] \cup \text{Star}(\text{chart}[i,j], \text{chart}[j,k]); \\
& \text{chart}[i,k] := \text{Closure}(\text{chart}[i,k]); \\
& \text{end} \\
& \text{end}; \\
& \text{if } S \in \text{chart}[0,n] \text{ then Accept else Reject}
\end{align*}$$

The algorithm presented above will be implemented into the following program where, due to some restrictions of the Pascal language, procedures for the three defined functions will be realized.

3. Implementing the algorithm

unit parser1;
interface
uses
Windows, Messages, SysUtils, Classes, Graphics,
Controls, Forms, Dialogs, StdCtrls, Grids, ExtCtrls;

type
TForm1 = class(TForm)
  StringGrid1: TStringGrid;
  Edit1: TEdit;
  Label1: TLabel;
  procedure Edit1KeyPress(Sender: TObject; var Key: Char);
  procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
Form1: TForm1;

implementation

{$R *.DFM}

const max=20; maxreg=20; maxcuv=5; maxcuvinte=20;

In order to memorize the elements displayed to the left and to the right of a grammar rule, we use the type Word.

type Cuvint=String[30];
  MultimeDeCuvinte=object
    nc: Integer;
    cuv: array[1..maxcuv] of Cuvint;
    procedure Adauga(c: Cuvint);
  end;
A rule is made up of two parts, one to the left and one to the right, and a grammar is made up of several of such rules.

```pascal
regula=record
    st,dr: Cuvint
end;

type Gramatica=object
    nr: Integer;
    reg: array[1..maxreg] of regula;
    procedure Citeste(nf: String);
end;
```

The diagram is a matrix where each cell is a set of words. G is the grammar with the rules and D is actually the word dictionary used.

```pascal
var chart:array[0..max, 0..max] of MultimeDeCuvinte;
G: Gramatica;
D: Gramatica;
Propoz: array[1..maxcuvinte] of Cuvint;
```

Drawing the diagram is done according to the content of the chart matrix by using the StringGrid1 control [3]:

```pascal
procedure DeseneazaChart(i,j: Integer);
var t: String; k,kk: Integer;
begin
    if chart[i,j].nc>=1 then
        begin
            t:='{
            for k:=1 to chart[i,j].nc-1 do
                t:=t+chart[i,j].cuv[k]+',
            t:=t+chart[i,j].cuv[chart[i,j].nc]+'}';
            Form1.StringGrid1.Cells[j,i+1]:=t;
            Form1.StringGrid1.Refresh
        end
end;
```

The following function checks whether a given word x is or is not inside a set of words M.

```pascal
function EsteIn(x: Cuvint; M: MultimeDeCuvinte): Boolean;
var este: Boolean; i: Integer;
begin
    este:=False;
    for i:=1 to M.nc do
        if x=M.cuv[i] then este:=True;
    EsteIn:=este
end;
```

For objects of the type MultimeDeCuvinte (SetOfWords) we have provided a procedure for adding a new word:

```pascal
procedure MultimeDeCuvinte.Adauga(c: Cuvint);
begin
    nc:=nc+1; cuv[nc]:=c
end;
```
Based on the last two subprograms described, we can define *Star*, *Closure* and *Lookup* operations, as well as the operation of reading a grammar whose rules are written in a given text file.

```pascal
procedure Star(X,Y: MultimeDeCuvinte;
var Z: MultimeDeCuvinte);
var i,j,k: Integer;
begin
  Z.nc:=0;
  for i:=1 to X.nc do
    for j:=1 to Y.nc do
      for k:=1 to G.nr do
        if G.reg[k].dr=X.cuv[i]+' '+Y.cuv[j] then
          Z.Adauga(G.reg[k].st)
end;

procedure Closure(S: MultimeDeCuvinte;
var C: MultimeDeCuvinte);
var i: Integer; gata: Boolean;
begin
  C:=S;
  repeat
    gata:=True;
    for i:=1 to G.nr do
      if EsteIn(G.reg[i].dr,C) then
        if not EsteIn(G.reg[i].st,C) then
          begin
            C.Adauga(G.reg[i].st);
            gata:=False
          end
  until gata
end;

procedure Gramatica.Citeste(nf: String);
var f: TextFile; s: String; p: Byte;
begin
  nr:=0;
  AssignFile(f,nf);
  Reset(f);
  while not eof(f) do
  begin
    nr:=nr+1;
    ReadLn(f,s);
    p:=Pos('->',s);
    reg[nr].st:=Copy(s,1,p-1);
    reg[nr].dr:=Copy(s,p+2,Length(s)-(p+1))
  end;
  CloseFile(f)
end;

procedure Lookup(k: Integer; var L: MultimeDeCuvinte);
var i: Integer;
begin
  L.nc:=0;
  for i:=1 to D.nr do
    if D.reg[i].dr=Propoz[k] then
      begin
        L.nc:=L.nc+1;
        L.cuv[L.nc]:=D.reg[i].st
      end
end;
```
Finally, the following procedure performs the grammatical analysis of the given phrase (ss), on the basis of the algorithm that has been theoretically described in the first paragraph.

```
procedure Parseaza(ss: String);
var i,j,k,kk: Integer;
n: Integer; p: Byte;
L,C,S: MultimeDeCuvinte; t: String;
begin
  ss:=ss+' ';
n:=0;
  while ss<>'' do
  begin
    p:=Pos(' ',ss);
n:=n+1;
    Propoz[n]:=Copy(ss,1,p-1);
    Form1.StringGrid1.Cells[n,0]:=Propoz[n];
    Str(n-1,t);
    Form1.StringGrid1.Cells[0,n]:=t;
    Delete(ss,1,p)
  end;
  Form1.StringGrid1.RowCount:=1+n;
  Form1.StringGrid1.ColCount:=1+n;
  Form1.StringGrid1.Show;
  for k:=1 to n do
  begin
    Lookup(k,L);
    Closure(L,C);
    chart[k-1,k]:=C;
    for i:=k-2 downto 0 do
    begin
      chart[i,k].nc:=0;
      for j:=k-1 downto i+1 do
      begin
        Star(chart[i,j],chart[j,k],S);
        for kk:=1 to S.nc do
        begin
          if not EsteIn(S.cuv[kk],chart[i,k]) then
            chart[i,k].Adauga(S.cuv[kk])
        end;
      end;
      Closure(chart[i,k],C);
    end;
  end;
  for i:=0 to n do
  begin
    for j:=0 to n do DeseneazaChart(i,j);
    if EsteIn('S',chart[0,n]) then
    begin
      Form1.Label1.Caption:= 'Fraza acceptata.'
    end;
    else
    begin
      Form1.Label1.Caption:= 'Fraza rejectata.'
    end;
  end;
end.
```

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  if Key=Chr(13) then Parseaza(Edit1.Text)
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Caption:='Basic CKY Parser'; StringGrid1.Hide;
  G.Citeste('gram.txt'); D.Citeste('lex.txt');
end;
```
Bellow are the contents of the two files (grammar and dictionary) that we have used in the examples we have analysed:

**GRAM.TXT**

<table>
<thead>
<tr>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>S-&gt;NP VP</td>
</tr>
<tr>
<td>NP-&gt;Pron</td>
</tr>
<tr>
<td>NP-&gt;N</td>
</tr>
<tr>
<td>NP-&gt;Det N</td>
</tr>
<tr>
<td>NP-&gt;NP AP</td>
</tr>
<tr>
<td>AP-&gt;A</td>
</tr>
<tr>
<td>AP-&gt;AP CP</td>
</tr>
<tr>
<td>CP-&gt;C A</td>
</tr>
<tr>
<td>VP-&gt;V VP</td>
</tr>
<tr>
<td>VP-&gt;V NP</td>
</tr>
</tbody>
</table>

**LEX.TXT**

<table>
<thead>
<tr>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>Det-&gt;orice</td>
</tr>
<tr>
<td>Det-&gt;fiecare</td>
</tr>
<tr>
<td>Det-&gt;o</td>
</tr>
<tr>
<td>Det-&gt;un</td>
</tr>
<tr>
<td>Pron-&gt;el</td>
</tr>
<tr>
<td>N-&gt;barbat</td>
</tr>
<tr>
<td>N-&gt;femeie</td>
</tr>
<tr>
<td>V-&gt;iubeste</td>
</tr>
<tr>
<td>V-&gt;uraste</td>
</tr>
<tr>
<td>A-&gt;frumoasa</td>
</tr>
<tr>
<td>A-&gt;desteapta</td>
</tr>
<tr>
<td>C-&gt;si</td>
</tr>
<tr>
<td>C-&gt;sau</td>
</tr>
</tbody>
</table>

We should mention the fact that the algorithm works only if there are at least two symbols in the right side of each grammar rule, therefore we should pay due attention to rewriting the grammars that we use, so that they may meet with this restriction.

**Conclusion**

We described in this paper the CKY algorithm that can be applied for lexical and syntactical analysis of Romanian sentences, if we write the grammar as a CNF grammar. Also, we implemented the algorithm in Delphi 3.0, developed by Borland (www.borland.com).

**References:**


