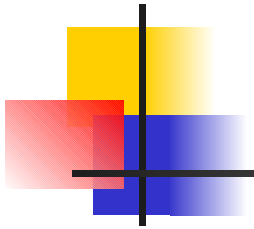
Evolution and Overgeneralization of Grammar

Shruti Dube

Y3337



Introduction

- Natural languages are characterized as a combination of rule-based generalization and lexical idiosyncracy
- The rules governing the syntax are actually quasi regular.
- Apparently allowed constructions are idiosycratically missing

Idiosyncracies

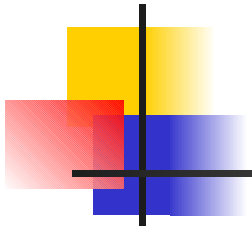
- Baker's Paradox

  The English past tense where the irregular form 'went' replaces '*goed*'

- Dative Shift:

  John gave/donated a book to the library

  John gave/*donated the library a book



- *'to be'* deletion rule

  The baby appears/seems to be happy

  The baby appears/seems happy

  The baby seems to be sleeping

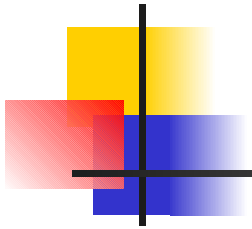
  *The baby appears/seems sleeping

- Lexical constraints

  strong/high/*stiff winds

  strong/*high/*stiff currents

  strong/*high/stiff breeze

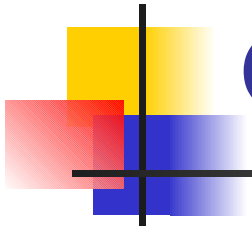


- Transitive/Intransitive

  John broke the cup
  The cup broke
  John kissed Mary
  *Mary kissed

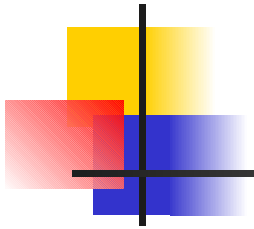
- We cannot interpret syntactic behaviour from semantics

  John waved Mary goodbye
  John waved goodbye to Mary
  *John said Mary hello
  John said hello to Mary

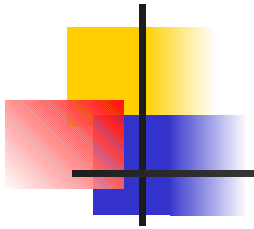


Question…

- Given these holes and quasi regular rules, what really is the learning mechanism ?

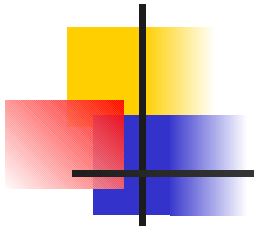


- Since only a finite set of sentences are heard, absence of a form does not imply that it is not allowed.
- Such holes are specific to languages so they cannot be attributed to postulating a Universal Grammar



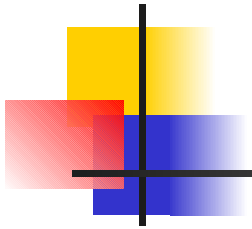
Approach

- Learning quasi regular structures in a rudimentary language from positive evidence alone using the Simplicity principle.
- As a measure for simplicity, MDL (Minimum Description Length) was used:Kolmogorov complexity



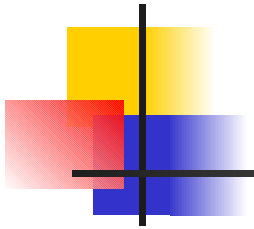
- Simplicity :Cognitive Systems prefer simpler patterns over complex ones. The shortest program that regenerates the object is a natural measure of its complexity
- Instead of actual binary coding, codelengths are calculated for this purpose.
- All grammars or hypotheses are expressed

$$C = C(H)+C(D|H)$$



Simulation

- 2 approaches :
  - There is a “super speaker” for all the listeners who knows the correct grammar entirely. All his utterances shall be completely correct and can have no exceptions
  - Transmission over generations: The listener listens to sentences spoken by a speaker, who himself was the listener some time ago.



- The first approach, though fair enough for simulation purposes, is unrealistic
- The second is closer to reality.

It can be thought to be a model wherein a generation of parents instruct their successors about the exceptions in language. Children also posit exceptions on listening to their utterances

Algorithmic details:

- Rudimentary toy language:

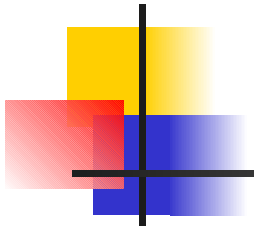
S1 = AB

S2 = BA

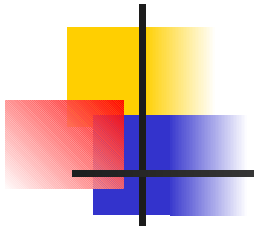
A={ a1,a2,a3,a4}

B= {b1,b2,b3,b4}

* = { (a1),(a2,b2),(a2,b3),(a2,b4),(b1,a1),
(b2,a1),(b3,a1),(b4,a1) }



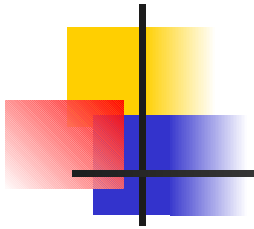
- Grammars are represented by matrices of probabilities and an exception matrix.
- Words are ranked according to their frequencies. The speaker and the listener share the probabilities of these words.
- Using Zipf's law the probabilities are calculated: $p = f/\sum f$



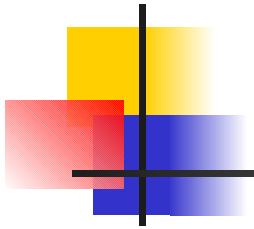
- Learning proceeds by gambles. The listener gambles whether a sentence can be an exception depending on whether it satisfies :

$$\text{Log}\,(1/p(x)\,) < N\,p(x)$$

- Scaling up of probabilities is done and if the gamble is correct, it results in shorter codelengths. If it is incorrect, the gamble is abandoned.



- Each time an exception is posited, a new hypothesis is generated
- A sentence which has been heard is never posited as an exception.
- The codelengths for these grammars are calculated.



- Each hypothesis is exposed to 50 sentences of a grammar, which in the first approach are from the perfect grammar, and in the second from the previous grammar.
- This attempts to mimic the situation of the poverty of stimulus, where one never hears all sentences and receive no negative feedback.

actual3

Class Edit Tools Options

Compile | Undo | Cut | Copy | Paste | Find... | Find Next | Close | Implementation

```java
import java.io.*;
import java.util.Random;


public class actual3
{

        static boolean[][] board;

        static grammar [] array = new grammar [32];

        static grammar actual3_grammar = new grammar();
        final static int [][] actual3_excep= new int [11][2];
        final static int [][] actualtobexposed=new int [32][2];
        final static
int[][]exceptions={{2,3},{4,3},{5,3},{7,3},{1,1},{1,2},{1,3},{2,1},{4,1},{5,1},{7,1}};

        static int excep_row=0;
        static int excep_col=0;
        static boolean [][] posited =new boolean[8][4];
        boolean [][] tried_for_positing= new boolean[8][4];

    actual3()
    {
     for (int i=0;i<8;i++)
        {
            for(int j=0;j<4;j++)
            {
                tried_for_positing[i][j]=false;
            }

        }
    }
    //takes as argument the grammar num and exceptions in prev grammar
    //returns num of excep in current grammar generated
```

Class compiled - no syntax errors

Class Edit Tools Options

Compile | Undo | Cut | Copy | Paste | Find... | Find Next | Close | Implementation

```
    }
    //Takes as argument the grammar rule and exceptions in prev grammar
    //returns one of excep in current grammar generated
    public int generates grammar(int gram_num,int excep_no)
    {

    InputStreamReader isr= new InputStreamReader (System.in);
BufferedReader br= new BufferedReader (isr);

        boolean except_chosen=true;
        int count=0;
        int v=0;
        int row;
        int col;
        int actual_from;
        int actual_to;
        int temcol;
        int [][] actio_exposed_to= new int [50][2];
        double prob_of_excep;
        double temp1=1;
        double temp2;
        double temp3;
        double temp4=0;
        double temp5;
        double temp6;
        double temp7=0;
        boolean once_done=false;
    boolean [][] once_calculated= new boolean [5][4];
    Random generator= new Random();
     for (int i=0;i<8;i++)
        {
            for(int j=0;j<4;j++)
            {
                once_calculated[i][j]=false;
            }
```

Class compiled - no syntax errors

Class Edit Tools Options

Compile | Undo | Cut | Copy | Paste | Find... | Find Next | Close | Implementation

```
        boolean once_done=false;
    boolean [][] once_calculated= new boolean [5][4];
    Random generator= new Random();
     for (int i=0;i<8;i++)
        {
            for(int j=0;j<4;j++)
            {
                once_calculated[i][j]=false;
            }

        }
    for (int i=0;i<5;i++)
        {
            for(int j=0;j<4;j++)
            {
                tried_for_posting[i][j]=false;
            }

        }

        for ( int i=0;i<30;i++)
        {
            do
            {

                actual3row = generator.nextInt( 8 );
                temrcol = generator.nextInt( 10);
                actual3col= array[gram_num+1].probac_choosing_matrix_actual[row][temrcol];
            }
            while(actual3 grammar.see word prob[actualrow][actualcol]==0);
            array[gram_num+1].indicheard[actual3row][actual3col]=true;

            sents_exposed_to [i][0]=actual3row;
            sents_exposed_to [i][1]=actual3_col;
```

Class Edit Tools Options

Compile | Undo | Cut | Copy | Paste | Find... | Find Next | Close | Implementation

```
    int loc;
    double *sum=0;
    double sec=1;
    double totsum=0;
    if (grow_num ==  _)
    {
        for (int _ =0;i<50;i++)
        {
            var= accts_exposed_to [i][0];
            loc = accts_exposed_to [i][1];
            ssum = array[0].sec_word_prob[var][loc]*array[0].fit_word_prob[var];
            sec = Math.pow(ssum,-1);
            totsum = totsum + Math.log(sec)/Math.log(2);
        }
        System.out.println("For the completely regular grammar codelength = "+totsum);
        return (0);
    }

       //generates a possible if a could be exception
       do
       {

        do
        {
             if ( except_exhausted[tried_for_positn])
             {
                 System.out.println("These are all the grammars that can be generated");

                 System.exit(0);
             }
            count++;

            row = generator.nextInt( 8 );

            temp=0 = generator.nextInt( 10);
```

Class Edit Tools Options

Compile | Undo | Cut | Copy | Paste | Find.. | Find Next | Close | Implementation

```
        cross_fire_pointing[row,[col]=true;

    }
    while(!array[gram_num].indicears[row][col]==true) | (posited[row][col]==true);
    proc_of_excep=(array_gram_num].tir_word_proc_row]*(array[gram_num,.occ_word_prob[row][co_];
    posited[row][col]=true;

    if (once_done==false)
    {

    for ( int i=0;i<50;i++)
    {
        {
    actual3row=events_exposed_to[i][0];
    actual3col=events_exposed_to[i][1];

    }
        array[gram_num+1].indicated[actual3row][actual3col]=true;

        once_done=true;
        actual3row=events_exposed_to [i][0];
        actual3col=events_exposed_to [i][1];
;
        if ((row==actual3row)&&(col==actual3col))
        {
            excep_closed=false;
    break;
        }
        temp1=(array_gram_num].tir_word_proc_row]*(array[gram_num,.occ_word_prob[row][co_];
        temp2=Math.pow(temp1,-1);
        temp3=Math.log(temp2)/Math.log(2);
        temp4=Math.pow((1-temp1),-1);
        if (i<((1.1)*temp_/temp3))
        {
            excep_closed=true;
```

Class Edit Tools Options

Compile | Undo | Cut | Copy | Paste | Find.. | Find Next | Close | Implementation

```
        }

    }
  }
c_cc
    {

        for(int i=0; i<50;i++)
        {
            actual3row=events_exposed_to[i][0];
            actual3col=events_exposed_to[i][1];
            if ((row==actual3row)&&(col==actual3col))
            {
                except_chosen=false;
                break;
            }
            temp1=(array_gram_num).tir_word_proc_row)*(array[gram_num].occ_word_prob[row][col]);
            temp2=Math.pow(temp1,-1);
            temp3=Math.log(temp2)/Math.log(2);
            temp4=Math.pow((1-temp1),-1);
            if (((1-1)*temp3)/temp4)
            {
                except_chosen=true;

            }

        }
    }

    if (except_chosen==true)
    {
      break;
    }

  }
```

Class Edit Tools Options

Compile | Undo | Cut | Copy | Paste | Find... | Find Next | Close | Implementation

```
        wcl_c(true);

          for(int i=0;i<8;i++)
          {
              for (int j=0;j<4;j++)
              {

                  array[gram_num+1].fir_word_prob[i]=array[gram_num].fir_word_prob[i];
                  array[gram_num+1].sec_word_prob[i][j]=array[gram_num].sec_word_prob[i][j];
                  array[gram_num+1].exceptcodelength=array[gram_num].exceptcodelength;

              }
          }

          array[gram_num+1].sec_word_prob[row][col]=0;

        array[gram_num+1].sec_word_prob[row][col]=0;


        for (int i=0;i<8;i++)
        {
            for (int j=0;j<4;j++)
            {
               if (!(array[gram_num-1].sec_word_prob[i][j] ==0)
                {
                    (array[gram_num+1].sec_word_prob[i][j])=((array[gram_num+1].sec_word_prob[i][j])*(Math.pow(temp4,0.5)));
                }
            }
        }


        for (int i=0;i<8;i++)
        {
            (array[gram_num-1].fir_word_prob[i])=((array[gram_num+1].fir_word_prob[i])*(Math.pow(temp4,0.5)));
```

Class Edit Tools Options

Compile | Undo | Cut | Copy | Paste | Find... | Find Next | Close | Implementation

```
int counter=0;
int x=0;
if (gram_num==0)
  {

      array[gram_num+1].list_of_excep_with_each_gram[i][0]=row;
      array[1].list_of_excep_with_each_gram[0][i]=col;


      for(int i=0; i<10;i++)
      {
          actual3row=cents_exposed_to[i][1];
          actual3col=cents_exposed_to[i][1];
          if (once_calculated[actual3row][actual3col]==true)
          {
              continue;
          }
          else
          {
              once_calculated[actual3row][actual3col]=true;
              temp1=Math.pow((array[gram_num+1].fix_word_prob[actual3row])*(array[gram_num+1].occ_word_prob[actual3row][actual3c

              temp2=(Math.log(temp1))/(Math.log(2));

              array[gram_num+1].datacodelength =array[gram_num+1].datacodelength+temp2;
          }


      }
  System.out.println("datacodelength for grammar no "+(gram_num+1) +" is "+array[gram_num+1].datacodelength);

  temp1=Math.pow((array[gram_num].fix_word_prob[row])*(array[gram_num].set_word_prob[row][col]),-1);
```

Class Edit Tools Options

Compile | Undo | Cut | Copy | Paste | Find... | Find Next | Close | Implementation

```
        temp2= Math.log(temp1)/(Math.log(2));

        array[gram_num+1].excepcodelength = array[gram_num+1].excepcodelength+temp2;
    System.out.println("excep code length =  "+ array[gram_num+1].excepcodelength);


    }


    while (array[gram_num].list_of_excep_with_each_gram[x][0] != -1)
    {
        array[gram_num+1].list_of_excep_with_each_gram[s][0]=array[gram_num].list_of_excep_with_each_gram[s]*0';
        array[gram_num+1].list_of_excep_with_each_gram[s][1]=array[gram_num].list_of_excep_with_each_gram[s]__;
        x++;


    }

    counter=0;
     if(gram_num!=0)
    {

        int exchange=counter;
        while(array[gram_num+1].list_of_excep_with_each_gram[counter][0]!= -1)
        {


           exchange=counter+1;
           for(int i=0; i<5; i++)
            {
                actual3too=events_exposed_to[i][0];
                actual3tol=events_exposed_to[i][1];

                if
(array[gram_num+1].list_of_excep_with_each_gram[counter][0]==actual3toxtra(array[gram_num+1].list_of_excep_with_each_gram[counter][1]
```

Class Edit Tools Options

Compile | Undo | Cut | Copy | Paste | Find... | Find Next | Close | Implementation

```
                array[gram_num+1].sec_word_prob[actual3row][actual3col]=array[0].sec_word_prob[actual3row][actual3col];
                temp5=Math.pow((array[0].fir_word_prob[actual3row])*(array[i].sec_word_prob[actual3row][actual3col]),(1/i));

                temp6=(Math.log(temp5))/(Math.log(2));
                array[gram_num+1].exceptodelength  =array[gram_num+1].exceptodelength-temp6;

                array[gram_num+1].list_of_excep_with_each_gram[counter][0]=array[gram_num-1].list_of_excep_with_each_gram[exch
                array[gram_num+1].list_of_excep_with_each_gram[counter][1]=array[gram_num-1].list_of_excep_with_each_gram[exch
                array[gram_num+1].list_of_excep_with_each_gram[exchangecounter][0]=-1;
                array[gram_num+1].list_of_excep_with_each_gram[exchangecounter][1]= 1;
                exchangecounter++;
                //scale down probabilities
                for (int j=0;j<5;j++)
                {
                    for (int k=0;k<4;k++)
                    {

                        array[gram_num+1].sec_word_prob[j][k]=(array[gram_num-1].sec_word_prob[j][k])*(Math.pow((1/(array[0].
                    }
                }
                for(int j=0;j<5;j++)
                {
                    array[gram_num+1].fir_word_prob[j]=array[gram_num-1].fir_word_prob[j]*Math.pow((1/array[0].fir_word_prob
                }


            }

        }
        counter++;
    }

    array[gram_num+1].list_of_excep_with_each_gram[counter][0]=row;
```

Class Edit Tools Options

Compile | Undo | Cut | Copy | Paste | Find... | Find Next | Close | Implementation

```
                array[gram_num+1].sec_word_prob[actual3row][actual3col]=array[0].sec_word_prob[actual3row][actual3col];
                temp5=Math.pow((array[0].fir_word_prob[actual3row])*(array[i].sec_word_prob[actual3row][actual3col]),(1/i));

                temp6=(Math.log(temp5))/(Math.log(2));
                array[gram_num+1].exceptodelength  =array[gram_num+1].exceptodelength-temp6;

                array[gram_num+1].list_of_excep_with_each_gram[counter][0]=array[gram_num-1].list_of_excep_with_each_gram[exch
                array[gram_num+1].list_of_excep_with_each_gram[counter][1]=array[gram_num-1].list_of_excep_with_each_gram[exch
                array[gram_num+1].list_of_excep_with_each_gram[exchangecounter][0]=-1;
                array[gram_num+1].list_of_excep_with_each_gram[exchangecounter][1]= 1;
                exchangecounter++;
                //scale down probabilities
                for (int j=0;j<5;j++)
                {
                    for (int k=0;k<4;k++)
                    {

                        array[gram_num+1].sec_word_prob[j][k]=(array[gram_num-1].sec_word_prob[j][k])*(Math.pow((1/(array[0].
                    }
                }
                for(int j=0;j<5;j++)
                {
                    array[gram_num+1].fir_word_prob[j]=array[gram_num-1].fir_word_prob[j]*Math.pow((1/array[0].fir_word_prob
                }


            }

        }
        counter++;
    }

    array[gram_num+1].list_of_excep_with_each_gram[counter][0]=row;
```

Class Edit Tools Options

Compile | Undo | Cut | Copy | Paste | Find... | Find Next | Close | Implementation

```
        }
        System.out.println("data code length for grammar number  "+(gram_num+1)+" is "+array[gram_num-1].datacodelength);


    temp1=Math.pow((array[gram_num].fix_word_prob[row]*(array[gram_num].sec_word_prob[row][col])),-1);
    temp2= Math.log(temp1)/(Math.log(2));
    array[gram_num].exceptcode_length = array[gram_num].exceptcode_length+temp2;
    System.out.println("exception code length for grammar number  "+(gram_num+1)+" is "+array[gram_num].exceptcodelength);

    double sum = array[gram_num-1].exceptcodelength+array[gram_num+1].datacodelength;
    System.out.println("the code length for grammar number "+(gram_num)+" is "+(sum);

  }

  System.out.println("this grammar contains " +(counter-1) +"exceptions");
  if (gram_num == 5 )
  {
      try
      {
          br.readline();
      }
      catch (IOException e){}
  }
  System.out.println();
  return (counter-1+1);

  }


public static void main(String args[])
{

   for(int i=0;i<32;i++)
```

Class Edit Tools Options

Compile | Undo | Cut | Copy | Paste | Find.. | Find Next | Close | Implementation

```
        actual3 grammar.sec_word_prob[7][3]=0;
        actual3_grammar.sec_word_prob[3][2]=0;
        actual3_grammar.sec_word_prob[1][2]=0;
        actual3_grammar.sec_word_prob[1][0]=0;
        actual3 grammar.sec_word_prob[6][2]=0;
        actual3_grammar.sec_word_prob[4][2]=0;
        actual3_grammar.sec_word_prob[5][2]=0;
       actual3_grammar.sec_word_prob[7][1]=1;
        actual3 obj= new actual3();
        int count=-1;
        int exceps=1;
         int b=1;
       while (true)
        {
            b = obj.generate_grammar(count,exceps);
            exceps++;
        count++;
        }
       //doing the grammar and the exceptions now
    }
    public boolean exce( exhausted[boolean[][] b)
    {
        for(int i=0;i<8;i ++)
        {
            for(int j=0;j<4;j++)
            {
                if ( b[i][j]==true)
                {
                    return false;
                }
            }
        }
        return true;
    }
}
```

BlueJ: Terminal Window - Evolution

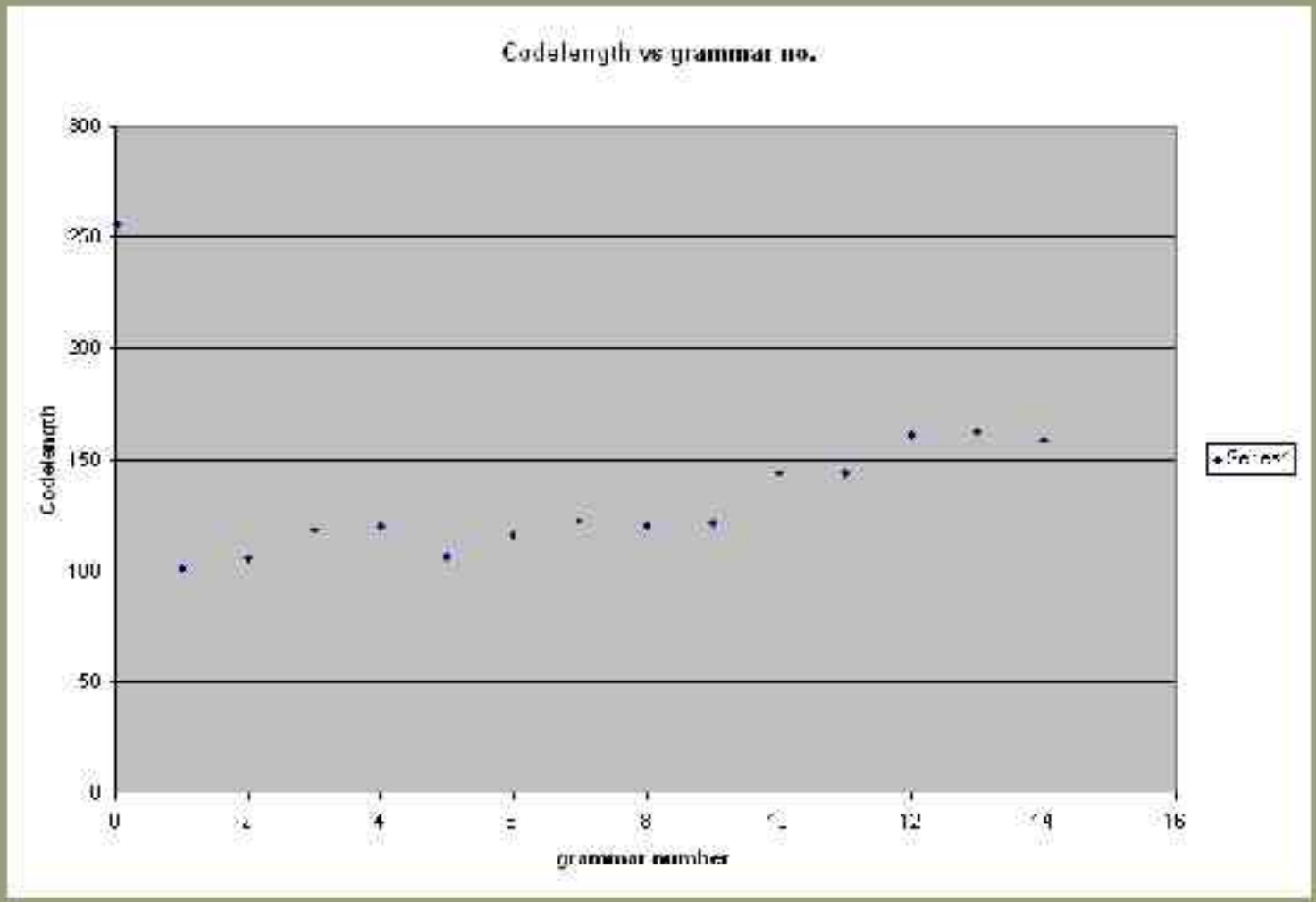
Options

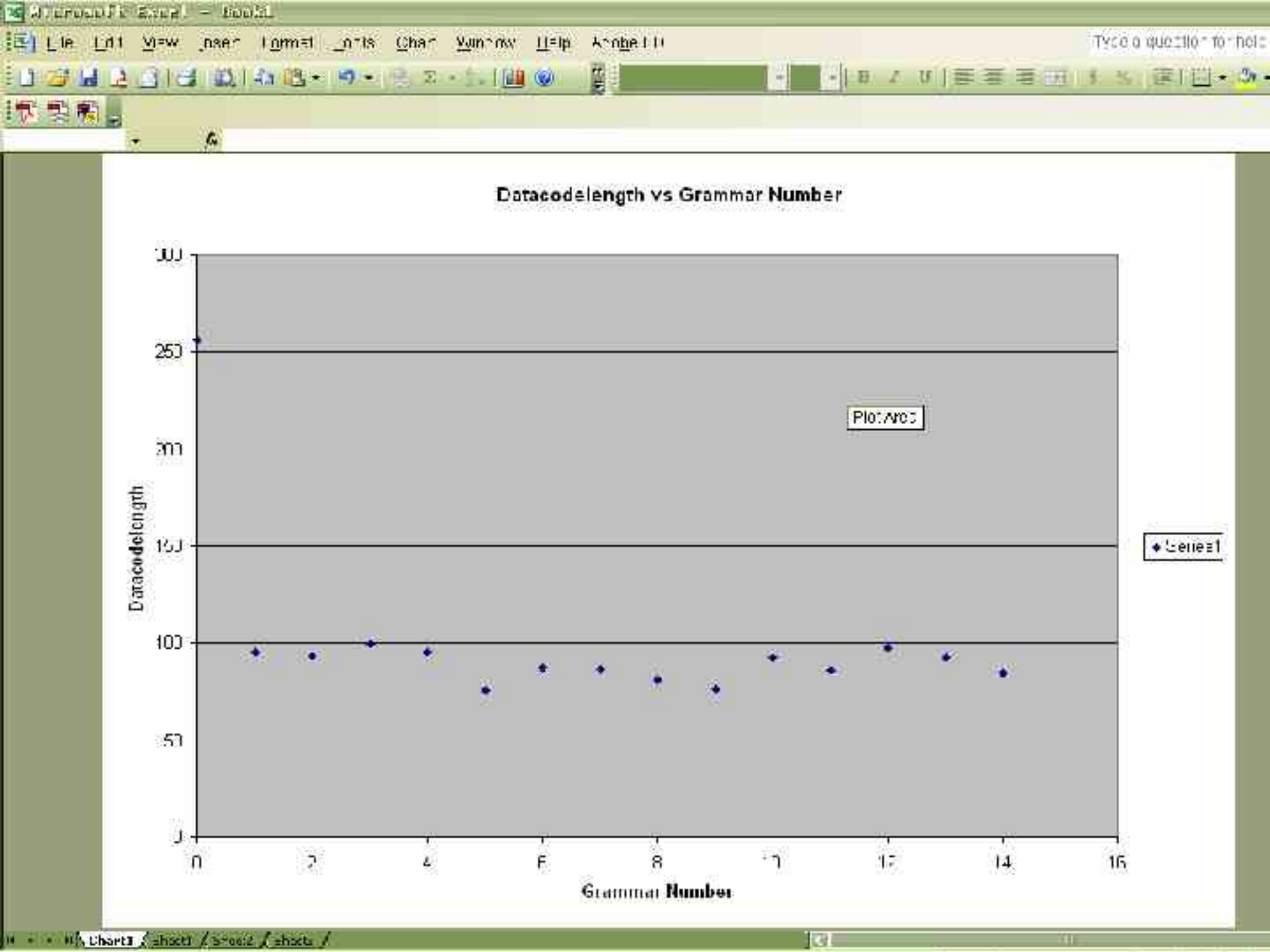
```
For the completely regular grammar codelength = 256.03436695302585
datacodelength for grammar no 1 is 55.72831780816085
excep code length = 5.186325815333204
This grammar contains 1exceptions

data code length for grammar number  2 is 93.72568554241206
exception code length for grammar number  2 is 11.451475536268673
The code length for grammar number 2 is 105.17716207770774
This grammar contains 2exceptions

data code length for grammar number  3 is 97.70572577794206
exception code length for grammar number  3 is 10.214051015590775
The code length for grammar number 3 is 107.92077209501738
This grammar contains 3exceptions

data code length for grammar number  4 is 95.07407972095654
exception code length for grammar number  4 is 24.641359181518028
The code length for grammar number 4 is 130.01543900551457
This grammar contains 4exceptions

data code length for grammar number  5 is 75.63553003391694
exception code length for grammar number  5 is 30.636763938224555
The code length for grammar number 5 is 106.27229297224149
This grammar contains 5exceptions

data code length for grammar number  6 is 87.20427359503941
exception code length for grammar number  6 is 28.523047727566125
The code length for grammar number 6 is 115.72732087269535
This grammar contains 5exceptions
```

BlueJ: Terminal Window - Evolution

Options

```
data code length for grammar number  7 is 86.36349831764398
exception code length for grammar number  7 is 36.8371324760857
The code length for grammar number 7 is 123.19960769232964
This grammar contains 6exceptions

data code length for grammar number  8 is 83.41327260887959
exception code length for grammar number  8 is 36.44011427490576
The code length for grammar number 8 is 119.85138582751555
This grammar contains 6exceptions

data code length for grammar number  9 is 76.33201375100579
exception code length for grammar number  9 is 45.42159217057597
The code length for grammar number 9 is 121.75399593746335
This grammar contains 7exceptions

data code length for grammar number  10 is 92.57351864183541
exception code length for grammar number  10 is 51.36354154345792
The code length for grammar number 10 is 143.93735048429758
This grammar contains 8exceptions

data code length for grammar number  11 is 85.93858943294727
exception code length for grammar number  11 is 57.87591854255876
The code length for grammar number 11 is 143.81450797965167
This grammar contains 9exceptions

data code length for grammar number  12 is 97.27678635060514
exception code length for grammar number  12 is 63.07441510057392
The code length for grammar number 12 is 160.35015195114108
This grammar contains 10exceptions

data code length for grammar number  13 is 92.90975451406511
exception code length for grammar number  13 is 69.66369419495495
The code length for grammar number 13 is 162.50874875583016
This grammar contains 11exceptions

data code length for grammar number  14 is 83.39950179048208
exception code length for grammar number  14 is 74.94584845321615
The code length for grammar number 14 is 158.34595672262823
This grammar contains 12exceptions
```


Codelength vs grammar no.
300
250
200
150
100
50
0
Codelength
0
4
8
12
16
grammar number


Datacodelength vs Grammar Number
Datacodelength
Grammar Number
PlotArea
Series1
0
2
4
6
8
10
12
14
16
50
100
150
200
250
300

```
For the completely regular grammar codelength = 257.71843506437986
datacodelength for grammar no 1 is 93.60852932947383
except codi length =  5.671288217054461
This grammar contains 1exceptions

data code length for grammar number  2 is 86.95137988534552
exception codi length for grammar number  2 is 12.158426511201108
The code length for grammar number 2 is 95.10572067966902
This grammar contains 2exceptions

data code length for grammar number  3 is 102.66127855336505
exception code length for grammar number  3 is 17.702560053130140
The code length for grammar number 3 is 120.44493345649654
This grammar contains 3exceptions

data code length for grammar number  4 is 95.40701050430071
exception code length for grammar number  4 is 24.600344321025714
The code length for grammar number 4 is 120.01816290541603
This grammar contains 4exceptions

data code length for grammar number  5 is 82.41085617967458
exception code length for grammar number  5 is 29.597523821965336
The code length for grammar number 5 is 112.0183779815409?
This grammar contains 5exceptions

data code length for grammar number  6 is 86.55875680495813
```

data code length for grammar number 7 is 92.239420655821S1
exception code length for grammar number 7 is 43.90342533053384
The code length for grammar number 7 is 136.14284588714844
This grammar contains 7exceptions

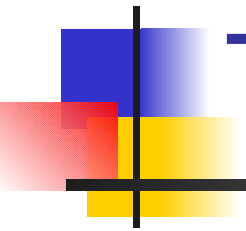
data code length for grammar number 8 is 78.16945049502652
exception code length for grammar number 8 is 49.14719482348606
The code length for grammar number 8 is 127.31664726171527
This grammar contains 8exceptions

data code length for grammar number 9 is 85.2254e+011587S8
exception code length for grammar number 9 is 55.70673644300706
The code length for grammar number 9 is 141.01227145885937
This grammar contains 9exceptions

data code length for grammar number 10 is 90.06917027071606
exception code length for grammar number 10 is 61.67497053165890
The code length for grammar number 10 is 155.54412903537756
This grammar contains 10exceptions

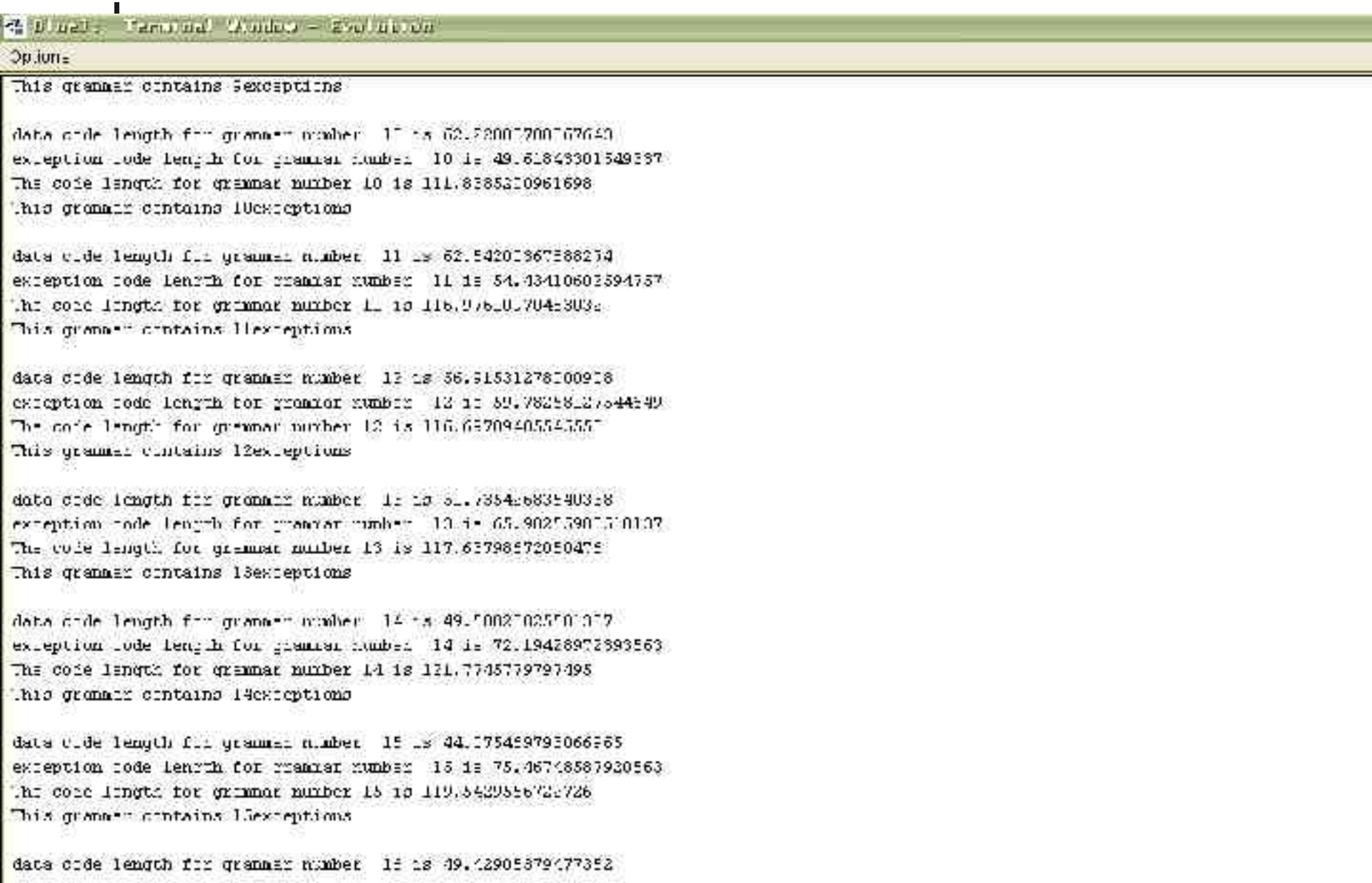
data code length for grammar number 11 is 102.81559238447402
exception code length for grammar number 11 is 63.95357516031135
The code length for grammar number 11 is 17...76015724473529
This grammar contains 11exceptions

These are all the grammars that can be generated

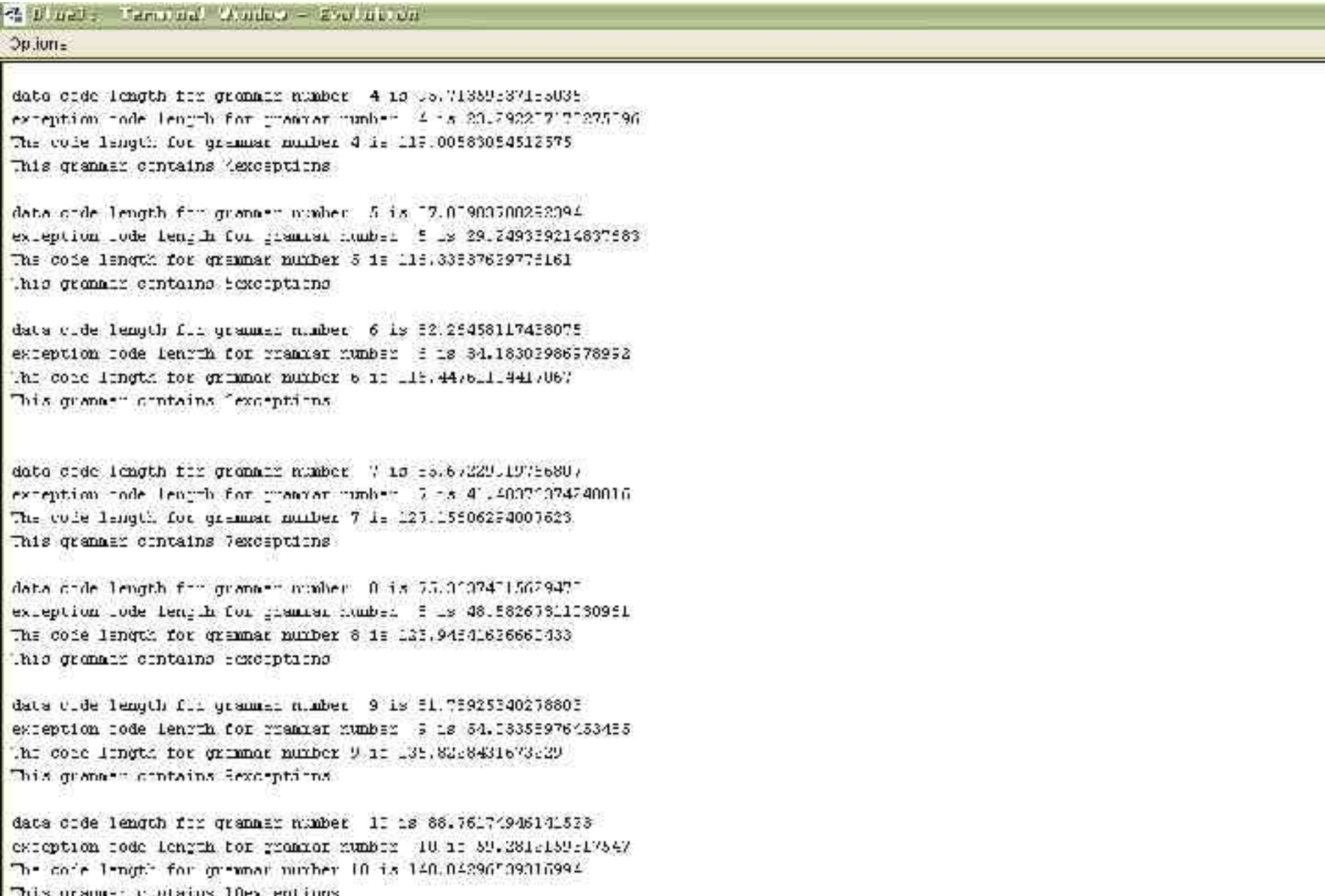


The Second Approach:-

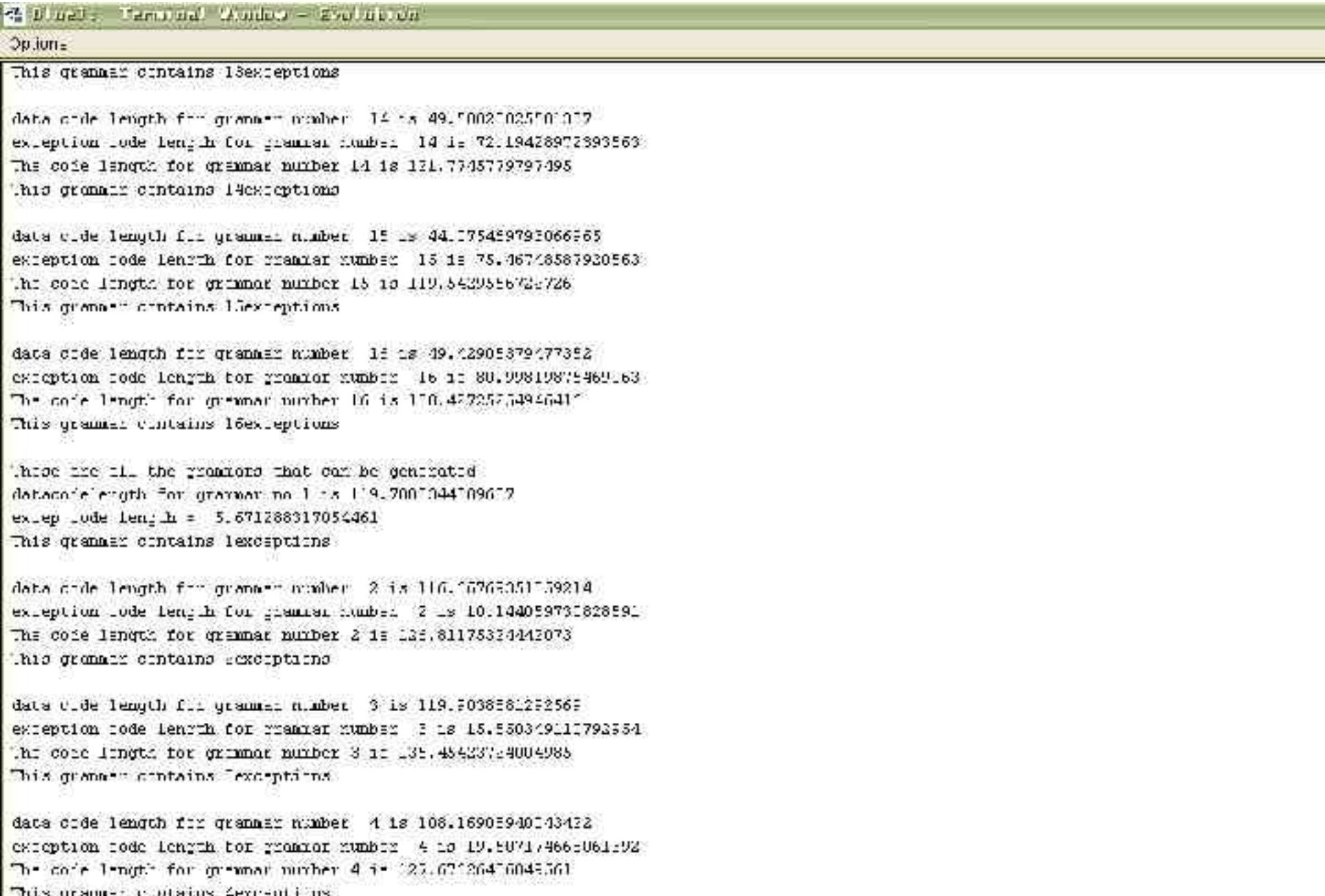
For 20 sentences:number of exceptions are very large



For 30 sentences: number of exceptions have decreased



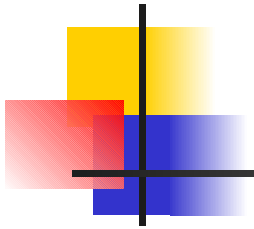
For 40 sentences : very few grammars are generated



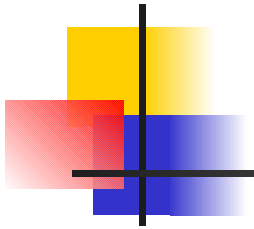
With a larger corpus of data: approach 1

Larger corpora : approach 2

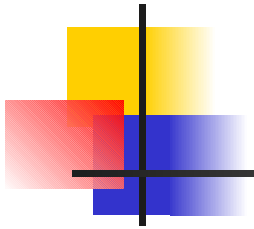
Conclusion:



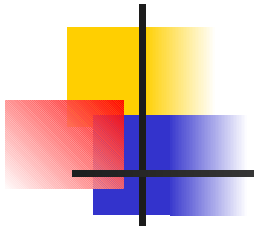
- When a small corpus of data was chosen, the simplicity principle yielded the exact number of 11 exceptions in most cases.
- The exceptions obtained were the same as those found in the actual language.
- So, for a small corpus of data, given the “super speaker approach”,the simplicity principle works quite efficiently.



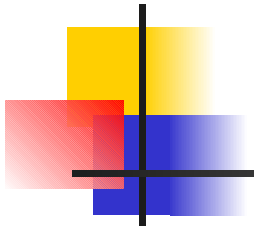
- However, if we look at the second approach. When we were using 50 sentences approximately 6 exceptions were found.
- If the number of sentences uttered are decreased, the exceptions possible increase.
- It is true that the datacodelegth decreases as we increase the number of exceptions. Since coding these exceptions requires some bits as well, whether that investment can be reobtained remains questionable.



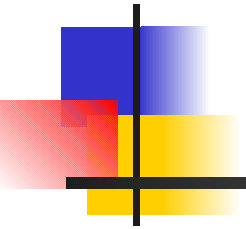
- When the number of sentences heard were approximately equal to the number of sample sentences, nearly perfect results were obtained.
- With the increased sample size however, this fails badly.
- The codelengths show a monotonic increase



- Even grammars with 11 exceptions never had the same exceptions as were present in the grammar.
- This happens primarily because data is chosen probabilistically. Since the exceptions themselves are randomly decided, investment in coding them may be quite large itself.



- Since this language has no semantics and no communicative function, it doesn’t model the relation between meaning-signal-referents.
- Though it is true for small corpora of sentences, it cannot be established as a general learning mechanism.



Thank you!