Asian School of Management And Technology

Gongabu, Kathmandu

Tribhuvan University

Compiler Design and Construction

Submitted By

Name: Anisha Lamichhane

Roll No: 20982

6th Semester Bsc.Csit

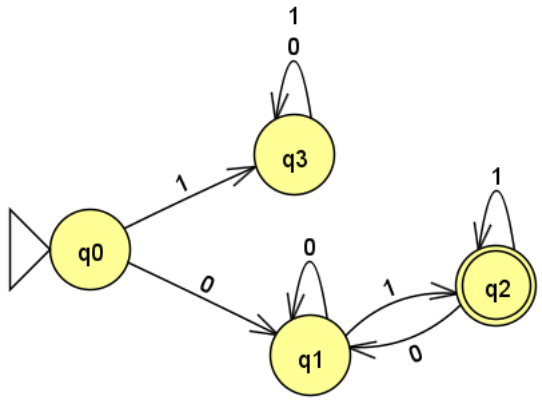
Submitted To

Name: Mr. Bikash Balami

Date: 2079/07/21

1. Design a DFA to simulate the following machines.

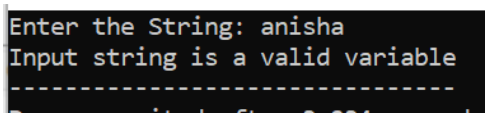
a. Accepting binary string that start with 0 and ends with 1.

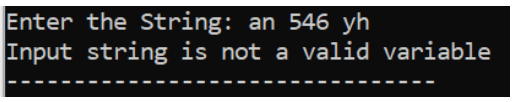


b. Accepting the valid variable names in C program

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main()
{
   char string[50];
   int count=0, i;
   printf("Enter the String: ");
   gets(string);
   if((string[0]>='a'&&string[0]<='z')   ||   (string[0]>='A'&&string[0]<='Z')   ||
(string[0]=='_'))
   {
      for(i=1;i<=strlen(string);i++)
      {
```

```
        if((string[i]>='a'&& string[i]<='z') || (string[i]>='A' && string[i]<='Z') || (string[i]>='0'&& string[i]<='9') || (string[i]=='_'))
        {
           count++;
        }
      }
   }
   if(count==(strlen(string)-1))
  {
      printf("Input string is a valid variable");
  }
  else
  {
     printf("Input string is not a valid variable");
   }
  return 0;
  }
```

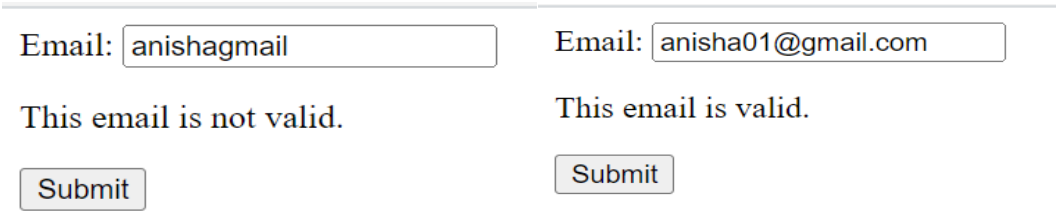




c. Accepting the valid gmail ID.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <label>Email:</label>
    <input type="text" id="emailId" /><br />
    <button onclick="validateGmail()">Submit</button>
    <script>
      function validateGmail() {
        var email = document.getElementById("emailId").value;
        regex =
          /^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-
]+)+$/;

        if (email.match(regex)) {
          alert("Valid Gmail");
        } else {
          alert("Invalid Gmail");
        }
      }
    </script>
  </body>
</html>
```



d. Accepting the prepaid NTC number

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Document</title>
 </head>
 <body>
  <label>Phone number:</label>
  <input type="text" id="num" /><br />
  <button onclick="validateNumber()">Submit</button>
  <script>
   function validateNumber() {
    var num = document.getElementById("num").value;
    regex = /^984\d{7}$/;

    if (num.match(regex)) {
     alert("Valid Ntc Prepaid number");
    } else {
     alert("Invalid Ntc Prepaid number");
    }
```

```
    }
  </script>
 </body>
</html>
```

Phone number: 9457537

This number is not valid

Submit

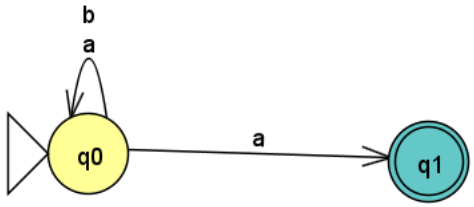
Phone number: 9841236786

This number is valid

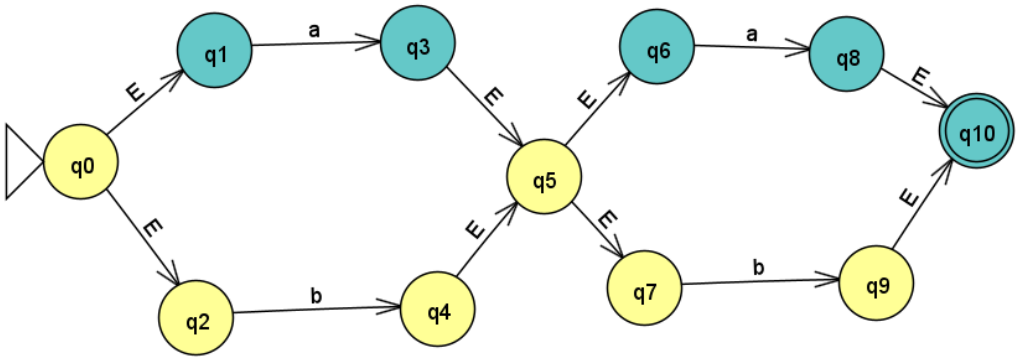
Submit

2. Design a NFA to simulate the following machine

a. Accepting RE (a + b)*a over = {a,b}



b. Accepting RE (a+b)(a+b) over = {a,b}.



3. Write a program to remove left recursion from the following grammar.

a. S ->Sab | ab | a | b (Grammar 1)

b. A ->A0 | A1 | 0 (Grammar 2)

```
gram = {}

def add(str):                          #to rules together
   str = str.replace(" ", "").replace("          ", "").replace("\n", "")
   x = str.split("->")
   y = x[1]
   x.pop()
   z = y.split("|")
   x.append(z)
   gram[x[0]]=x[1]

def removeDirectLR(gramA, A):
        """gramA is dictonary"""
        temp = gramA[A]
        tempCr = []
        tempInCr = []
        for i in temp:
                if i[0] == A:
                        #tempInCr.append(i[1:])
                        tempInCr.append(i[1:]+[A+"'"])
                else:
```

```
                    #tempCr.append(i)
                    tempCr.append(i+[A+"'"])
        tempInCr.append(["|e"])
        gramA[A] = tempCr
        gramA[A+"'"] = tempInCr
        return gramA


def checkForIndirect(gramA, a, ai):
        if ai not in gramA:
                return False
        if a == ai:
                return True
        for i in gramA[ai]:
                if i[0] == ai:
                        return False
                if i[0] in gramA:
                        return checkForIndirect(gramA, a, i[0])
        return False

def rep(gramA, A):
        temp = gramA[A]
        newTemp = []
        for i in temp:
                if checkForIndirect(gramA, A, i[0]):
                        t = []
                        for k in gramA[i[0]]:
                                t=[]
```

```
                        t+=k
                        t+=i[1:]
                        newTemp.append(t)
                else:
                        newTemp.append(i)
        gramA[A] = newTemp
        return gramA

def rem(gram):
        c = 1
        conv = {}
        gramA = {}
        revconv = {}
        for j in gram:
                conv[j] = "A"+str(c)
                gramA["A"+str(c)] = []
                c+=1
        for i in gram:
                for j in gram[i]:
                        temp = []
                        for k in j:
                                if k in conv:
                                        temp.append(conv[k])
                                else:
                                        temp.append(k)
                        gramA[conv[i]].append(temp)
        #print(gramA)
        for i in range(c-1,0,-1):
```

```
        ai = "A"+str(i)
        for j in range(0,i):
                aj = gramA[ai][0][0]
                if ai!=aj :
                        if aj in gramA and checkForIndirect(gramA,ai,aj):
                                gramA = rep(gramA, ai)

for i in range(1,c):
        ai = "A"+str(i)
        for j in gramA[ai]:
                if ai==j[0]:
                        gramA = removeDirectLR(gramA, ai)
                        break

op = {}
for i in gramA:
        a = str(i)
        for j in conv:
                a = a.replace(conv[j],j)
        revconv[i] = a

for i in gramA:
        l = []
        for j in gramA[i]:
                k = []
                for m in j:
                        if m in revconv:
                                k.append(m.replace(m,revconv[m]))
```

```
                                    else:
                                        k.append(m)
                                l.append(k)
                            op[revconv[i]] = l

                return op

n = int(input("Enter No of Production: "))
for i in range(n):
    txt=input()
    add(txt)
result = rem(gram)
for x,y in result.items():
    print(f'{x} -> ', end="")
    for index, i in enumerate(y):
        for j in i:
            print(j, end="")
            if (index != len(y) - 1):
                print("", end="")
    print()
```

```
Enter No of Production: 1
S -> Sab| ab | a | b
S -> abS'aS'bS'
S' -> abS'|e
```

```
D:\Users\User\PycharmProjects\pythonPro
Enter No of Production: 1
A -> A0 | A1 | 0
A -> 0A'
A' -> 0A'1A'|e
```

4. Write a program to create a symbol table for the variables (for data types only)

program.txt

```
int a = 9 ;
float b = 6.79 ;
```

Python code

```
tokens=[' ']
ids = []
key_words = ['int','string','char','float','double']
operators = ['+','-','*','/','=','<']
punct = ['(',')', '{', '}' , '[' ,']',',']
with open("program.txt") as t:
   a=t.readlines()
   for t in a:
      tokens=tokens + (t.split(" "))
print("id\t\t\tdata_types\t\tvalue")
for pos, t in enumerate(tokens):
   for k in key_words:
      if(t==k):
         ids.append(tokens[pos + 1])
         if(tokens[pos + 2 ] == ','):
            print(tokens[pos + 1] + " " + tokens[pos] + "      " + "NULL")
            tokens.insert(pos + 3, tokens[pos])
         elif (tokens[pos + 2] =='('):
            end=tokens.index(')')
            para = tokens[pos + 3:end]
```

```
            kc=0
            pt=[]
            for key in key_words:
                kc = kc + para.count(key)
                i=0
                while(i < para.count(key)):
                    pt.append(key)
                    i = i + 1
                print(tokens[pos + 1] + "\t\t\t" + tokens[pos] + "\t\t\t" + str(kc) + "\t\t\t" + str(pt))
        elif (tokens[pos + 1] == '('):
                continue
        Else:
                print(tokens[pos + 1] + "\t\t\t" + tokens[pos] + "\t\t\t" + tokens[pos + 3])
```

```
D:\Users\User\PycharmProjects\pythonProject\venv
id          data_types      value
a           int         9
b           float           6.79

Process finished with exit code 0
```

5. Write a program to find set of non-terminals, set of terminals, set of productions and starting symbol. Here you have to take the CFG as input in file.

grammar.txt

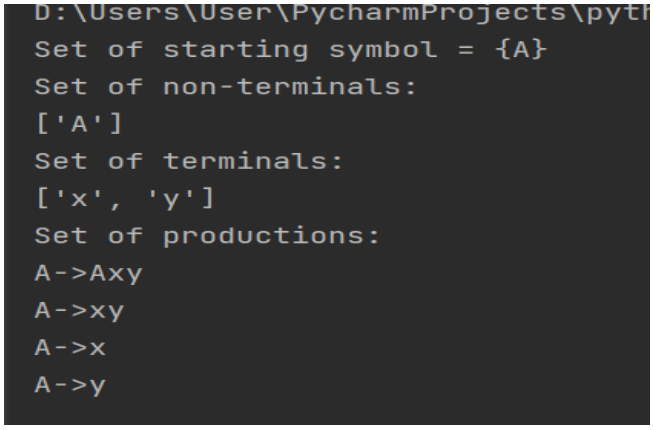
A->Axy|xy|x|y

Python code

```
import re
tokens=[]
with open("grammar.txt") as t:
    a=t.readlines()
    for t in a:
        tokens=tokens + (t.split("->"))

    r = re.findall('([A-Z])', tokens[1]) #get all capital letters from rhs
    rSmall = re.findall('([a-z])', tokens[1]) #get all small letters from rhs
    rSmallUpdated  = list(dict.fromkeys(rSmall)) #remove duplicates

    productions = tokens[1].split("|")
    print("Set of starting symbol = {"+tokens[0]+"}")
    print("Set of non-terminals:")
    print(r,sep=",")
    print("Set of terminals:")
    print(rSmallUpdated,sep=',')
    print("Set of productions:")
    for item in productions:
        print(tokens[0]+"->"+item)
```



6. Write a program to realize the concept of loop optimization in compiler optimization in following cases, with respect to running time only.

a. Code Motion

```
#include<stdio.h>

#include <time.h>

void program1(){

   int a,b,c,d,e,f,g,h =0;

   for(int i=0; i<1000000000; i++) {

      c = d + e;

      f = g + h;

      a = b + i ;
```

```
    }
}
int main(){
    clock_t t;
    t = clock();
    program1();
    t = clock() - t;
    double time_taken = ((double)t)/CLOCKS_PER_SEC; // in seconds


    printf("program1() took %f seconds to execute \n", time_taken);
}
```

```
program1() took 3.550000 seconds to execute

--------------------------------
```

Program 2(After reducing frequency)

```
void program2(){
    int a,b,c,d,e,f,g,h =0;
    c = d + e;
    f = g + h;
    for(int i=0; i<1000000000; i++) {
        a = b + i ;
    }
}
```

```
program2() took 3.511000 seconds to execute

--------------------------------
```

b. Loop Jamming

Loop jamming or loop fusion is a technique for combining two similar loops into

one, thus reducing loop overhead by a factor of 2. For example, the following

C code:

```
for (i=1;i<=100;i++)
  x[i]=y[i]*8;


for (i=1;i<=100;i++)
  z[i]=x[i]*y[i];
```

can be replaced by

```
for (i=1;i<=100;i++)
{
 x[i]=y[i]*8;
    z[i]=x[i]*y[i];
};
```

Code:

```
#include<stdio.h>
#include <time.h>

void loopJam(){
    int a =1;
    int b= 5;
    for(int i =0;i<1000000000;i++){
        a = a + i;
    }
```

```
    for(int j =0;j<1000000000;j++){
        b = b + j;
    }
}
int main(){
    clock_t t;
    t = clock();
    loopJam();
    t = clock() - t;
    double time_taken = ((double)t)/CLOCKS_PER_SEC; // in seconds

    printf("loopJam() took %f seconds to execute \n", time_taken);
}
```

```
loopJam() took 5.299000 seconds to execute

--------------------------------
```

After reducing:

```
#include<stdio.h>
#include <time.h>
void loopJam(){
    int a =1;
    int b= 5;
    for(int i =0;i<1000000000;i++){
        a = a + i;
        b = b + i;
    }
}
```

```
int main(){
    clock_t t;
    t = clock();
    loopJam();
    t = clock() - t;
    double time_taken = ((double)t)/CLOCKS_PER_SEC; // in seconds
    printf("loopJam() took %f seconds to execute \n", time_taken);
}
```