Projeto
Teoria da Computação

Lucas B. Fialho – 2712
Wisney Bernardes - 1285
Marciley Oliveira - 2768

Projeto

- Minimização
- GLC->Chomsky

Metodologia

- Linguagem de Programação

Metodologia

- Projeto 1
  - Dividido em três classes:
    - AFD
    - INOUT
    - MINIMIZADOR

Metodologia

- Projeto 1
  - AFD
  - Responsavel pela representação das transiçoes e as operações por meio de grafo.

P1 - AFD

```
class AFD:
    def __init__(self,alfabeto,estados,inicial,final):
        self.alfabeto=[]#tipo lista
        for i in range(0,len(alfabeto)):
            self.alfabeto.append(str(alfabeto[i]))
        self.estados=[]#tipo lista
        for i in range(0,len(estados)):
            self.estados.append(str(estados[i]))
        self.transicao={}#tipo dict para melhor representação de grafos
        self.inicial=str(inicial)#tipo string representa o estado inicial
        self.final=[]#tipo lista representa todos os estados finais do AFD
        for i in range(0,len(final)):
            self.final.append(str(final[i]))
    #função de transição de um estado lendo um terminal para outro estado
    def ftransicao(self,e1,ch,e2):
        if(e1 in self.transicao):
            trans=self.transicao[str(e1)]
            trans[str(ch)]=str(e2)
        else:
            self.transicao[e1]={str(ch):str(e2)}
    def reconhece(self,palavra):
        estado=self.inicial
        out={}
        for i in range(0,len(palavra)):

            if(palavra[i] not in self.alfabeto):
                out={'status':False,'motivo':PALAVRA_FORA_DO_ALFABETO}
                return out

            elif(estado not in self.transicao or palavra[i] not in self.transicao[estado]):
                out={'status':False,'motivo':PALAVRA_COM_TRANSICAO_INVALIDA}
                return out
            else:
                estado=self.transicao[estado][palavra[i]]
        if estado not in self.final:

            out={'status':False,'motivo':PALAVRA_NAO_CHEGOU_NO_ESTADO_FINAL}
```

Metodologia

- Projeto 1
  - Minimizador
  - Classe Responsavel por operações de minimização seguindo todas etapas

P1 – Def. Minimizador

```
def __init__(self,afd):
    self.afd=afd#AFD para ser minimizado
    self.table={}#tabela de minimização
    self.lista={}#lista auxiliar de estados pendentes
```

P1 – Transições Indefinidas

```
def transicao_indefinida(self):
    for estado in self.afd.estados:
        for ltr in self.afd.alfabeto:
            if estado not in self.afd.transicao or ltr not in self.afd.transicao[estado]:
                if 'd' not in self.afd.estados:
                    self.afd.estados.append('d')
                self.afd.ftransicao(estado,ltr,'d')
                self.afd.ftransicao('d',ltr,'d')
    return self
```

P1 – Estados Inuteis

```
def estado_inacessivel(self):
    for i in range(0,len(self.afd.estados)):
        estado=self.afd.estados[i]
        status=False
        if estado == self.afd.inicial:
            status=True
            continue

        for e in self.afd.transicao:#percorre todas as transiçoes.

            for ltr in self.afd.alfabeto:#percorre o alfabeto
                if self.afd.transicao[e][ltr]==estado:#verifica se existe transição dado o estado e e um terminal
                    status=True#SE EXISTE ENTÃO É TRIVIAL PERCORRER TODO ALFABETO.
                    break
        if status==False:#se o status for igual a false significa que o estado i não é alcançavel.
            del self.afd.transicao[self.afd.estados[i]]#remove as transições do estado inacessivel.
            del self.afd.estados[i]#remove o estado inalcançavel
            self.estado_inacessivel()#verifica se existe mais estados inacessiveis[CHAMADA RECURSIVA]
            break
    return self
```

P1 – Preenche Tabela

```
def preenche_tabela(self):
    self.table={}
    for i in range(0,len(self.afd.estados)):
        for j in range(0,i):
            if ((self.afd.estados[j] in self.afd.final and  self.afd.estados[i] not in self.afd.final)
             or (self.afd.estados[i] in self.afd.final and  self.afd.estados[j] not in self.afd.final)):
                #marcar
                res='X'
            else:
                #não marcar
                res=None
            if self.afd.estados[i] in self.table:
                aux=self.table[self.afd.estados[i]]
                aux[self.afd.estados[j]]=res
            else:
                self.table[self.afd.estados[i]]={self.afd.estados[j]:res}
    return self
'''
```

P1 – Equivalencia de estados não marcados

```
def verifica_estados_nao_marcados(self):
    for i in self.table:
        for j in self.table[i]:
            if self.table[i][j]==None:
                self.checa_equivalencia(i,j)
```

```
def checa_equivalencia(self,e1,e2):
    i=0
    for ch in self.afd.alfabeto:
        if self.afd.transicao[e1][ch] == self.afd.transicao[e2][ch]:
            i=i+1

        elif (self.afd.transicao[e1][ch] in self.table
            and self.afd.transicao[e2][ch] in self.table[self.afd.transicao[e1][ch]]):
            if self.table[self.afd.transicao[e1][ch]][self.afd.transicao[e2][ch]]=='X':

                self.table[e1][e2]='X'
                if str(self.afd.transicao[e1][ch])+','+str(self.afd.transicao[e2][ch]) in self.lista:
                    #entao vamos marcar todos da lista.

                    for tail in self.lista[str(self.afd.transicao[e1][ch])+','+str(self.afd.transicao[e2][ch])]:

                        estado1=tail.split(',')[0].strip()
                        estado2=tail.split(',')[1].strip()
                        self.table[estado1][estado2]='X'
                    del(self.lista[str(self.afd.transicao[e1][ch])+','+str(self.afd.transicao[e2][ch])])


            else:
                if str(self.afd.transicao[e1][ch])+','+str(self.afd.transicao[e2][ch]) not in self.lista:
                    self.lista[str(self.afd.transicao[e1][ch])+','+str(self.afd.transicao[e2][ch])]=[str(e1)+','+str
                else:
                    self.lista[str(self.afd.transicao[e1][ch])+','+str(self.afd.transicao[e2][ch])].append(str(e1)+'
```

P1 – Junta Estados não marcados

```
def joinStates(self):

    #duplica a instancia da classe minimizador sendo assim só mexemos com o objeto da classe copiada.
    newAfd=copy.copy(self.afd)
    for i in self.table:
        for j in self.table[i]:
            if self.table[i][j]==None:
                #junta os estados e remove os dois antigos transformando em unico com todas as transiçoes.
                newAfd.estados.append(str(j)+str(i))

                if j not in self.afd.transicao:
                    transicaoj={}
                else:
                    transicaoj=self.afd.transicao[j]
                if i not in self.afd.transicao:
                    transicaoi={}
                else:
                    transicaoi=self.afd.transicao[i]
                newAfd.transicao[str(j)+str(i)]=self.union(transicaoi,(transicaoj))
                '''AJUSTA AGORA TODOS AS TRANSIÇÕES ANTIGAS PARA O NOVO ESTADO'''
                for estado_atual in newAfd.transicao:
                    for letra in newAfd.transicao[estado_atual]:
                        if i in newAfd.transicao[estado_atual][letra] or j in newAfd.transicao[estado_atual][letra]:
                            newAfd.transicao[estado_atual][letra]=str(j)+str(i)
                if i == newAfd.inicial or j == newAfd.inicial:
                    newAfd.inicial=str(j)+str(i)
                if i in newAfd.final or j in newAfd.final:
                    if i in newAfd.final:
                        del(newAfd.final[newAfd.final.index(i)])
                    if j in newAfd.final:
                        del(newAfd.final[newAfd.final.index(j)])
                    newAfd.final.append(str(j)+str(i))


                if i in newAfd.transicao:
                    del(newAfd.transicao[i])
                    estado_rem=newAfd.estados.index(i)
                    del(newAfd.estados[estado_rem])
                if j in newAfd.transicao:

                    del(newAfd.transicao[j])
                    estado_rem=newAfd.estados.index(j)
                    del(newAfd.estados[estado_rem])

                continue

            else:
                #apenas copia.
                pass
```

P1 – Metodo Principal

```
def minimiza(self):
    #CRIA ESTADO D PARA TODAS AS TRANSIÇÕES INDEFINIDAS
    #REMOVE TODOS OS ESTADOS INACESSIVEIS APARTIR DO ESTADO INICIAL
    #Preenche a tabela de estados com dados iniciais.
    self.transicao_indefinida().estado_inacessivel().preenche_tabela()
    #Agora temos que analizar os estados que não estão marcados(None)
    self.verifica_estados_nao_marcados()


    self.afd=self.joinStates()
    self.transicao_indefinida()

    return self.afd
```

P1 - INOUT

```
class INOUT:
    def __init__(self,fileIN,fileOUT):...
    #é feita a leitura linha a linha do arquivo de entrada
    def readFile(self):...
    #escrita no arquivo de saida do AFD minimizado.
    def save(self,afd):...
```

Projeto 2

- GLC-Simplificada X Chomsky

Metodologia

- Projeto 2

```
# -*- coding: utf-8 -*-
import random
import string
import copy
import sys
class Gramatica:...
class Chomsky:...
class INOUT:...
if len(sys.argv)<2:
    print('Entradas apenas por terminal!!!')
i=INOUT(sys.argv[1],sys.argv[2])
g=i.readFile()
c=Chomsky(g)
print('Gramatica não normalizada \n %s\n'%g.p)
g=c.converte()
i.save(g)
print('Gramatica normalizada por Chomsky: \n %s\n'%g.p)
```

P2 - Gramatica

```
class Gramatica:
    def __init__(self,variaveis,terminais,inicial):
        self.variaveis=list(variaveis)
        self.terminais=list(terminais)
        self.inicial=str(inicial)
        self.p={}
    def regraProducao(self,variavel,alpha):
        if variavel in self.p:
            prod=self.p[variavel]
            prod.append(alpha)
        else:
            self.p[variavel]=[alpha]
```

P2 – Chomsky - PASSO 1

```
class Chomsky:
    def __init__(self,gramatica):
        self.g=gramatica
    def converte(self):
        nGramatica=copy.copy(self.g)
        for v in self.g.variaveis:
            if v not in self.g.p:
                continue
            for p in self.g.p[v]:
                contaVar=[]
                contaTer=[]
                for i,s in enumerate(p):
                    if s in self.g.variaveis:
                        contaVar.append([i,s])
                    elif s in self.g.terminais:
                        contaTer.append([i,s])

                if len(contaTer)!=0 and len(contaVar)!=0:

                    for terPivo in contaTer:


                        while True:
                            newVar=random.choice(string.ascii_uppercase)
                            if newVar not in self.g.variaveis:
                                self.g.variaveis.append(newVar)
                                break

                        if terPivo[0]==len(p):
                            string_nova=(p[0:terPivo[0]])+str(newVar)

                        else:
                            string_nova=(p[0:terPivo[0]])+str(newVar)+p[terPivo[0]+1:]


                        self.g.regraProducao(newVar,terPivo[1])
```

P2 – Chomsky – PASSO 2

```
                    if p in self.g.p[v]:
                        self.g.p[v][self.g.p[v].index(p)]=string_nova

                    p=string_nova
        #pega todas as produções com tres variaveis e reduz para duas.
        for v in self.g.variaveis:
            if v not in self.g.p:
                continue
            for i,p in enumerate(self.g.p[v]):
                while (len(p)>2):

                    while True:
                        newVar=random.choice(string.ascii_uppercase)
                        if newVar not in self.g.variaveis:
                            self.g.variaveis.append(newVar)
                            break
                    self.g.regraProducao(newVar,p[1:])
                    self.g.p[v][i]=p[0]+newVar
                    p=self.g.p[newVar][0]

        return self.g
```

P2 – INOUT

```
class INOUT:
    def __init__(self,fileIN,fileOUT):...

    def readFile(self):
        self.fileIN=open(self.fileInputName,'r')
        gramatica=Gramatica([],[],'')
        while True:
            linha=self.fileIN.readline()
            if (linha.split('#')[0].strip())=='GLC':
                status=True
                break
            elif (linha.split('#')[0].strip()) not in [' ','',None]:
                status=False
                break
        if status==False:
            return
        #VERIFICO QUANTAS VARIAVEIS TEREMOS.
        while True:
            linha=self.fileIN.readline()
            if (linha.split('#')[0].strip()) not in [' ','',None,'0']:
                variaveis=range(0,int(linha.split('#')[0].strip()))
                break
        #leio as variaveis do arquivo.
        for v in variaveis:
            while True:
                linha=self.fileIN.readline()
                if (linha.split('#')[0].strip()) not in [' ','',None,'0']:
                    gramatica.variaveis.append(str(linha.split('#')[0].strip()))
                    break
        #leio quantos terminais teremos.
        while True:
            linha=self.fileIN.readline()
            if (linha.split('#')[0].strip()) not in [' ','',None,'0']:
                terminais=range(0,int(linha.split('#')[0].strip()))
                break
```

P2 – INOUT

```
        #preencho a lista de terminais.
        for t in terminais:
            while True:
                linha=self.fileIN.readline()
                if (linha.split('#')[0].strip()) not in [' ','',None]:
                    gramatica.terminais.append(str(linha.split('#')[0].strip()))
                    break
        #percorro o arquivo ate terminar pegando as produções
        while True:
            try:
                #primeiro pegamos uma variavel.
                while True:
                    linha=self.fileIN.readline()
                    if not linha: break
                    if (linha.split('#')[0].strip()) not in [' ','',None]:
                        variavel=(str(linha.split('#')[0].strip()))
                        break

                #pegamos o lado direito da produção.
                while True:
                    linha=self.fileIN.readline()
                    if not linha: break
                    if (linha.split('#')[0].strip()) not in [' ','',None]:
                        alpha=(str(linha.split('#')[0].strip()))
                        break
                if not linha: break
                #monto uma produção.
                gramatica.regraProducao(variavel,alpha)
            except EOFError, e:
                break
        self.fileIN.close()
        #self.fileOUT.close()
        gramatica.inicial=gramatica.variaveis[0]
        return gramatica
```

P2 – INOUT

```
def save(self,gramatica):
    self.fileOUT=open(self.fileOutputName,'w')
    self.fileOUT.write("GLC\t# identifica o tipo de formalismo\n")
    self.fileOUT.write(str(len(gramatica.variaveis))+"\t# quantidade de variaveis\n")
    for v in gramatica.variaveis:
        self.fileOUT.write(str(v)+'\n')
    self.fileOUT.write(str(len(gramatica.terminais))+"\t# quantidade de símbolos terminais\n")
    for t in gramatica.terminais:
        self.fileOUT.write(str(t)+'\n')
    self.fileOUT.write('# Listagem de Regras de Produção\n')
    for lado_esquerdo in gramatica.p:
        for i,lado_direito in enumerate(lado_esquerdo):

            self.fileOUT.write(str(lado_esquerdo)+'\n')
            self.fileOUT.write(str(gramatica.p[lado_esquerdo][i])+'\t #'+str(lado_esquerdo)+'->'+str(gramatica.p[



    self.fileOUT.close()
```